MASTER EN SCIENCES INFORMATIQUES

Eléments d'intelligence artificielle appliqués au monde de la communication

Verdeyen, Philippe

Award date:
2003

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année Académique 2002-2003

Eléments d'intelligence artificielle appliqués au monde de la communication

Philippe Verdeyen

Mémoire présenté en vue de l'obtention du grade de Licencié en Informatique.

Remerciements

Monsieur Jean-Marie Jacquet:

Pour l'opportunité offerte par ce mémoire.

Pour toute l'énergie consacrée à l'encadrement de ce mémoire.

Monsieur Olivier Mertens:

Pour toutes les remarques constructives.

Pour la lecture approfondie du mémoire.

Pour l'amélioration du kit de formation.

Pour les conseils relatifs à la structure du document.

Monsieur Pierre Vanpee

Pour toutes les remarques constructives.

Pour la lecture approfondie du mémoire.

Pour l'amélioration du kit de formation.

Docteur Marie-Alexandra Lambot

Pour la lecture du mémoire avec l'œil néophyte du médecin.

Pour la traduction anglaise de l'avant-propos.

Monsieur Mathys Merlijn

Pour la traduction néerlandaise de l'avant-propos.

Pour l'amélioration du kit de formation.

Pour leur participation active dans la mise au point du kit de formation, un tout grand merci à:

Jérémie Bury, Xavier Miller, Dave Oltenfreiter et Wim Waerenborgh

Les différents participants aux séances de formation, au nombre de 26 en date du premier août 2003.

Toute ma gratitude à l'entreprise GigaSpaces (http://www.gigaspaces.com) pour avoir fourni gratuitement une licence académique pour le produit : 'GigaSpaces ™' et pour l'efficacité de son support.

Table des figures

Avant-propos

L'information est devenue le moteur de notre société post-industrielle. Chaque jour, un flux toujours grandissant de courriers inondent nos boîtes postales électroniques. La capacité de décryptage d'un être humain est dépassée par l'abondance d'informations.

Ce mémoire propose une boîte à outils permettant le transfert vers la machine d'une partie de l'effort requis par le décodage de l'information. De par la nature évolutive de l'information, la boîte à outils est construite sur une théorie ensembliste permettant son extension plus aisée.

La boîte à outils se propose de modéliser l'information et ses acteurs. De même que la gestion des documents de type formulaire et une première approche de gestion de textes libres. L'ensemble serait incomplet sans la possibilité d'automatiser une succession de tâches. La modélisation d'une telle dynamique se fait au moyen d'un graphe de flux.

La technologie utilisée restant encore très confidentielle, ce travail contient aussi un support de formation se composant d'une courte présentation et d'exemples fonctionnels de programmation. L'ensemble est construit de telle manière qu'un public averti puisse recevoir une information pertinente en deux heures de présentation.

Une application concrète clôture ce mémoire permettant la validation du modèle choisi et une démonstration d'utilisation de la boîte à outils.

Foreword

Information has become the driving force of our post-industrial society. Every day an ever-growing mass of mail floods both our conventional and electronic mail boxes. The sheer amount of the information given overwhelms our human capacity to assimilate it.

This thesis offers a framework allowing the transfer of some of the decoding effort to the computer. Due to the evolutionary nature of information media, this framework is built on a ensemblist theory that will allow for easier extension.

The framework is based on the modelisation of information and its actors as well as the management of documents such as printed form and the first steps to manage free text. In this approach automatisation of a succession of tasks is mandatory. This modelisation is obtained by the use of a flowchart.

Since the technology we use here is still not widly used, we added to this work a formation support including a short presentation and functional programation exemples. It is designed so that an man of experience could be given an information to the point in two hours.

We end this work with a concret application allowing us to validate the chosen model and to demonstrate the use of the framework.

Voorwoord

Informatie is de motor van onze postindustriële samenleving. Elke dag overspoeld een steeds aangroeiende stroom van berichten onze elektronische postbus. Deze overvloed van gegevens overschrijdt het menselijke vermogen tot verwerking.

Deze thesis biedt een raamwerk aan dat toelaat om een deel van de gegevensverwerking door te schuiven naar de computer. Wegens de veranderlijke aard van de informatiemaatschappij is dit raamwerk gegrondvest op een theorie van verzamelingen hetgeen uitbreidingen makkelijker toelaat.

Het raamwerk bestaat uit de modellering van data en zijn spelers alsook het beheer van formulieren en een eerste benadering voor het beheer van vrije teksten. Voor de gevolgde aanpak is de mogelijkheid tot automatisering van opeenvolgende taken onmisbaar. De modellering van zulk een dynamiek is verwezenlijkt met behulp van een stroomschema.

Daar de gebruikte technologie weinig kenbaar is omvat dit werk het nodige vormingsmateriaal bestaande uit een korte beschrijving en voorbeelden van uitvoerbare programma's. Dit alles is zodanig opgebouwd dat een persoon met enige ervaring in het domein in twee uur een pertinente vorming kan krijgen.

Deze thesis wordt afgesloten met een concrete toepassing ter validatie van het gekozen model. Eveneens demonstreert dit het gebruik van het raamwerk.

Glossaire

Acteur
Voir 'Monde de la communication'.

Connaissance
Donnée contenue dans un ensemble de type Zone décrivant un fait.

Espace
Container à informations regroupées en Zone.

Monde de la communication
Composé d'acteurs consommant de l'informations diffusées par différents supports de transmission.

Ontologie
Matérialisation sous forme de spécification d'un concept abstrait.

Support de communication
voir 'Monde de la communication'.

Zone
Container à informations matérialisées sous formes de tuples de même dimension (le nombre d'éléments les composant).

1. Introduction

1.1. Objectifs

La genèse de ce mémoire est construite sur une seule question : 'Comment conceptualiser une secrétaire virtuelle de l'informations électroniques ?'. La conception d'un secrétariat virtuel ne peut se faire de l'intérieur, il n'est pas concevable de spécifier un ensemble de services sans comprendre les utilisateurs et les travaux à mener à bien !

Le choix de la technologie pour développer un secrétariat virtuel s'est porté sur la philosophie Linda et les technologies associées (comme JavaSpaces). JavaSpaces a été choisi par son aspect plus commercial (implémentation Java) que Linda, et surtout par le potentiel d'évolution de la technologie. Comparé à Linda, JavaSpaces offre des fonctionnalités supplémentaires: architecture supportant une forte connectivité, aspect de sécurisation et un fonctionnement basé sur la distribution de code. Mais cette comparaison avantageuse n'interdit pas un oeil critique sur la technologie. (voir les remarques en Annexe 3).

Le choix de l'utilisation de la technologie JavaSpaces impose un nouvel objectif à ce mémoire. Sa nature confidentielle impose la disponibilité d'un outil pédagogique (voir Annexe 3). Des séminaires ont déjà été dispensés auprès de professionnels de l'informatique et déjà de futurs séminaires sont planifiés.

1.2. Préparation

Pour conceptualiser un outil d'aide à la gestion, il est nécessaire de débuter par la lecture de documents concernant le sujet abordé. L'acquisition d'une philosophie de travail spécifique au domaine étudié en est le but. Dans la masse de la documentation scientifique disponible, aucun article n'a montré une corrélation suffisante avec le sujet du mémoire pour en justifier une synthèse. L'étude du sujet commence donc sur une base totalement originale.

Le second point de préparation est l'acquisition des connaissances utiles à l'utilisation des technologies Linda. Ces connaissances regroupent les concepts de programmation parallèle, de la philosophie Linda et de la technologie JavaSpaces.

1.3. Rédaction

Les cas d'utilisations :

La première phase est la compréhension des enjeux du monde de la communication. Pour cela, le domaine d'exploration voulu pour le secrétariat virtuel est synthétisé dans des cas d'utilisations. Les cas d'utilisations sont tous développés à partir de documents ou d'expériences réels.

Les cas d'utilisations développés sont :

- Le courrier électronique justifiée par l'obligation de tout système d'aide à la gestion de pouvoir au minimum représenter la réalité. Le cas étudié est un échange de courrier électronique.

- Les dynamiques d'échanges et d'acquisitions d'informations.

- Des factures (électricité et téléphone) et des formulaires d'inscriptions, qui sont des documents consultés pour la conception du cas d'utilisation des formulaires.

- La recherche de la présence d'une information spécifique dans un texte libre.

Structure de données

Les structures de données nécessaires au stockage et à l'exploitation de l'information sont conceptualisées en se basant sur les exigences énoncées dans les cas d'utilisations. Le formalisme utilisé est la notation ensembliste qui est le seul à pouvoir s'adapter à la volatilité de la structure de l'information. D'autres notations ont été évaluées mais sans pouvoir apporter une souplesse d'évolution acceptable, comme c'est le cas pour les graphes de taxinomie.

Services

Les différents services (primitives) mis à disposition par le secrétariat virtuel sont spécifiés en se basant sur les structures de données.

L'un des services disponibles dans le secrétariat virtuel est l'analyseur sémantique. Son rôle est de détecter la présence d'une sémantique dans un texte donné. Etant donné l'extrême complexité du sujet, son développement est limité car ne faisant pas l'objet de ce mémoire. L'objectif annoncé de l'analyseur est de proposer une architecture évolutive permettant l'intégration d'une langue autre que le français et une extension aisée à la manipulation de différentes structures de phrases.

Exemple final

Un cas d'utilisation transversal (regroupant les différents cas d'utilisation) a été rédigé dès le début de la préparation du mémoire. Lors de la conclusion de celui-ci, le cas d'utilisation est implémenté avec l'aide des primitives disponibles.

La méthode de travail utilisée valide le modèle des primitives et y apporte certaines améliorations. Ce dernier exemple est aussi source d'informations quant à l'utilisation en situation réelle d'une solution à base de technologie JavaSpaces. Des tests de performances avec le secrétariat virtuel ont été effectués à la demande de l'entreprise GigaSpaces, ce qui a permis l'amélioration des performances des espaces persistés dans la version 3 du produit. GigaSpaces est une implémentation commerciale de JavaSpaces.

2. Etude de cas du monde de la communication

L'objectif de ce mémoire est la conception d'un système évolutif permettant la maîtrise du flux de l'information généré dans le monde de la communication. Ce monde complexe dans une société sur-consommatrice est composé d'acteurs fournissant de l'information (les expéditeurs), d'acteurs la consommant (les destinataires) et repose sur différents supports de transmission (service postal, courrier électronique,...).

La finalité du système est la simulation de certains comportements humains dans le traitement de l'information. Sa modélisation reposera sur la notion même de conception de l'utilisateur plutôt que ses exigences, rendant le système d'aide à la décision complexe.

Le point de départ pour appréhender le monde de la communication consiste dans le développement de quelques systèmes simples dans le but de mettre en exergue les facteurs au postulat commun.

Les objectifs de cette première étape sont :

- fournir un premier modèle du monde de la communication
- proposer certaines solutions
- comprendre les défis à relever

2.1. Le dialogue électronique

2.1.1. Introduction

Le premier exemple illustre le cas du dialogue électronique qui utilise un support de communication unique: le courrier électronique. Les acteurs prenant part à ce dialogue peuvent être identifiés par une adresse électronique. Le dialogue débute avec la recherche par un étudiant d'une définition pour le mot: ontologie.

2.1.2. Scénario

Les trois acteurs impliqués dans le dialogue sont :

Référence	Description
`Etud@Universite.be`	Un étudiant
`Prof@Universite.be`	Un enseignant
`Ami@company.com`	Une relation de Etud

Etud@Universite.be prépare la présentation d'un examen, en relisant ses notes il ne retrouve plus la définition correcte du mot 'ontologie'. Pour obtenir de l'aide, sa première réaction est de s'adresser directement au professeur en charge du cours.

Etud@Universite.be ayant obtenu une définition, il désire trouver une illustration contextuelle de la définition; pour cela, il contacte un ami.

Ami@company.com donne la référence d'un site Internet (style encyclopédique).

Voici les différents messages électroniques échangés entre les acteurs. Par souci de concision, le contenu des messages ne reprend que les éléments indispensables à la bonne compréhension du scénario, en supprimant entre autre, toutes formules de politesses.

Les messages sont numérotés par ordre chronologique d'apparition.

message1			
de	Etud@fundp.be	**pour**	Prof@fundp.be
titre	besoin d'aide		
texte	Qu'est-ce une ontologie ?		

message2			
de	Prof@fundp.be	**pour**	Etud@fundp.be
titre	Re: besoin d'aide		
texte	Simple analytique de l'entendement pur. > > qu'est-ce une ontologie ?		

message3			
de	Etud@fundp.be	**pour**	Ami@company.com
titre	Fwd: Re: besoin d'aide		
texte	Comment illustrer cette définition ? > simple analytique de l'entendement pur. >> >> qu'est-ce une ontologie ?		

message4			
de	Ami@company.com	**pour**	Etud@fundp.be
titre	Illustration de l'utilisation 'ontologie'		
Texte	Regarde toujours sur www.encyclopédie.fr…		

Un observateur qui analyse le déroulement du dialogue peut établir des faits sur les acteurs, leurs relations, les sujets abordés,... Le dialogue permet d'affirmer que *Prof@Universite.be* et *Ami@company.com* sont connus de *Etud@Universite.be*, mais ne permet pas de conclure à une relation existante entre *Prof@Universite.be* et *Ami@company.com* puisque ces deux acteurs n'ont pas conversé directement (dans l'état actuel de nos connaissances).

Les connaissances déduites par ce dialogue peuvent se représenter à l'aide d'un graphe orienté,

les notations utilisées se retrouvent en [Annexe 1.1]. Pour chaque étape du dialogue, les éléments remarquables sont identifiés.

2.1.4. Après réception du message1

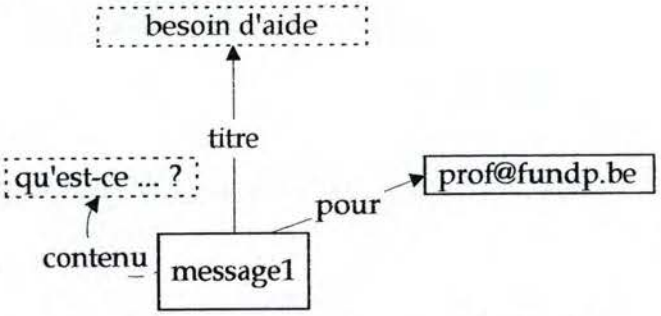


figure 1 : Etat des connaissances après message1

La réception du message1 produit un certain nombre d'entités (p.ex. `'etud@fundp.be'`). Les composants du message peuvent se retrouver au moyens des différents liens sémantiques ajoutés au graphes (titre, de, pour, ...).
Le graphe nous permet de répondre à certaines questions comme : 'Qui a envoyé le message1 ?'

2.1.5. Après réception du message2

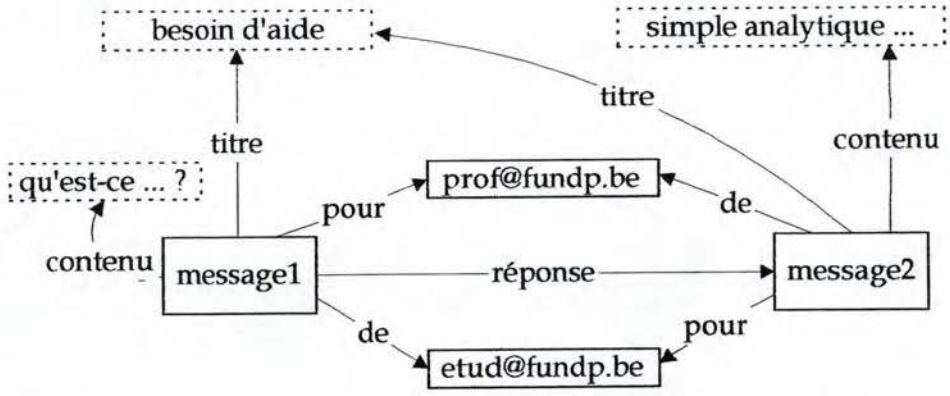


figure 2 : Etat des connaissances après message2

Le lien sémantique liant message1 et message2 est induit par les faits suivants :

- L'expéditeur du message1 est le destinataire du message2
- L'expéditeur du message2 est le destinataire du message1
- Le titre du message2 est une concaténation entre 'Re : ' et le titre du message1.

2.1.6. Après réception du message4

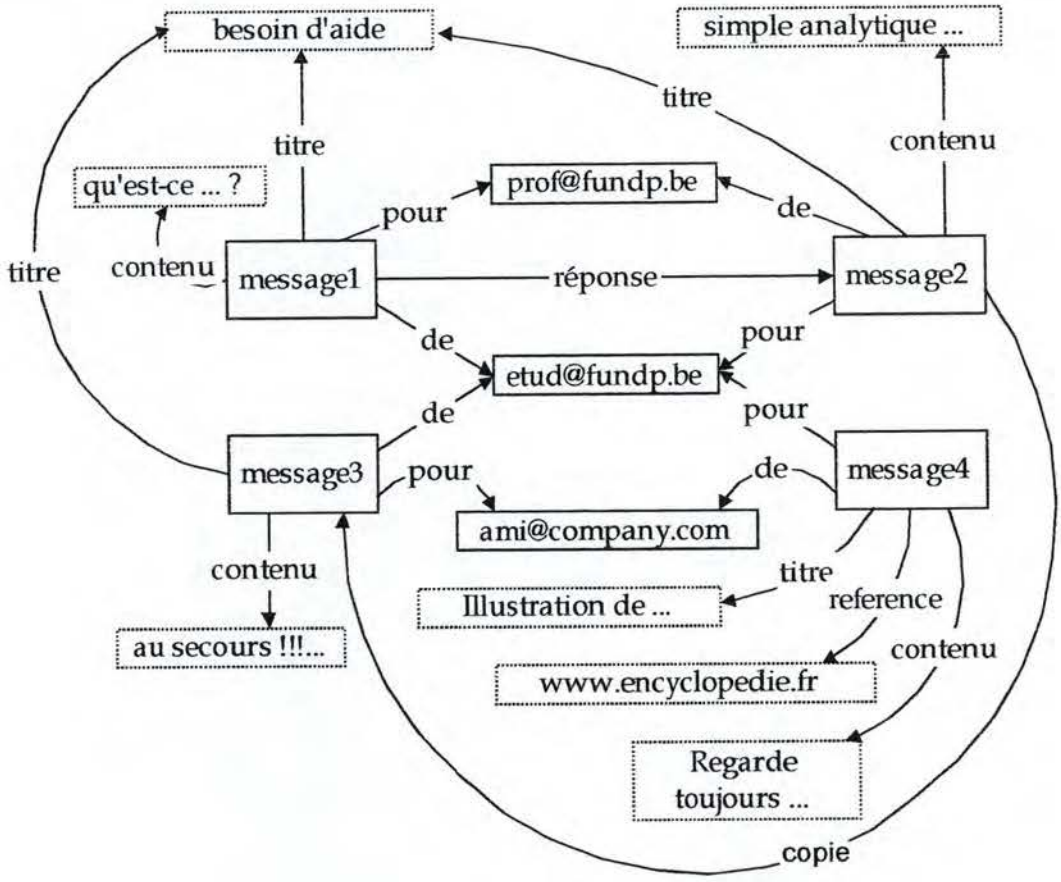


figure 3 : Etat des connaissances après message3 et 4

Le lien sémantique liant message2 et message3 est induit par les faits suivants :

- Le destinataire du message2 est l'expéditeur du message3
- Le titre du message3 est un concaténation entre 'Fwd : ' et le titre du message2

Remarque : Les définitions attribuées aux relations 'réponse' ou 'copie' ne permettent pas de reconnaître toutes les dépendances entre les messages. Pour cela, il suffit à un acteur de modifier l'intitulé du titre d'un message. Il n'existe aucun standard dans les échanges électroniques pour construire avec plus de rigueur un système d'identification de dépendances entre les messages.

2.2. Le dialogue administratif (flux de documents)

2.2.1. Introduction

Le second exemple décrit un échange de documents administratifs lors de l'inscription d'étudiants dans une université. La communication utilise un support papier (services postaux) ou une ligne téléphonique. La réception des messages peut provoquer l'envoi d'une réponse ou l'exécution d'un acte de gestion.

Les acteurs ne peuvent plus être identifiés par une adresse électronique, mais à l'aide d'une adresse postale ou un numéro de téléphone.

2.2.2. Scénario

Les différents acteurs prenant part à ce scénario sont :

Administration 44, rue du parc Bureaux 4 0458/65.23.25	Un représentant de l'administration universitaire.
Secrétariat 44, rue du parc Bureau 35	Un représentant de l'administration de la faculté Fac.
Professeur 46, rue du parc Bureau 60	Un professeur de la faculté Fac, chargé d'un cours en seconde année.
Etudiant1 64, rue des lilas	Un étudiant inscrit au sein de la faculté Fac, en première année.
Etudiant2 12, avenue du champs	Un étudiant inscrit au sein de la faculté Fac, en seconde année.
Etudiant3 35, rue du foulque	Un étudiant libre inscrit au sein de la faculté Fac, en première année.

Fixons arbitrairement une date comme attribut représentant le moment de réception de chaque document.

4 août 2001 — Un courrier est reçu contenant la demande d'inscription en première année de *Etudiant1*. La demande d'inscription est complète et valide. Un bulletin de virement est envoyé à *Etudiant1*.

19 août 2001 — L'agence bancaire confirme le paiement du minerval de *Etudiant1*.

21 août 2001 — *Etudiant2* contacte par téléphone l'*Administration* en vue d'obtenir la liste des documents utiles à son inscription en seconde année. Suite aux informations fournies par *Etudiant2*, le processus d'inscription ne pourra continuer qu'après réception d'un document d'équivalence. Un document d'équivalence est requis puisque *Etudiant2* est diplômé d'une école étrangère.

24 août 2001 — Le document d'équivalence d'*Etudiant2* est reçu par l'*Administration*. Selon les critères légaux, *Etudiant2* ne pourra prétendre qu'à une inscription en tant qu'élève libre. Un bulletin de virement adapté en conséquence est envoyé à *Etudiant2*.

29 août 2001 — Un courrier est reçu par l'*Administration* contenant la demande d'inscription en première année de *Etudiant3*. La demande d'inscription est validée, un bulletin de virement est envoyé à *Etudiant3*.

24 septembre 2001 — L'agence bancaire avertit l'*Administration* du paiement du minerval de *Etudiant2*, il devient dès lors définitivement inscrit avec un statut d'élève libre.

Extrait du règlement des inscriptions

Art.1 : La date limite du paiement d'un minerval est fixée au 31 octobre de l'année en cours, tout étudiant n'étant pas en ordre de paiement à cette date ne saurait prétendre légalement à une inscription.

Art.2 : Tout étudiant étranger désirant s'inscrire devra fournir une attestation d'équivalence. Le document d'équivalence A permet l'inscription de l'étudiant, tandis que le document d'équivalence B ne permet qu'une inscription au titre d'élève libre.

Ce scénario contient un flux de messages et un règlement de base dans lequel la dynamique de l'acquisition de connaissances est illustrée en fonction de trois profils disponibles :

- Administration représente les instances administratives de l'Université et est responsable de l'application des règles de gestion.
- Secrétariat représente les instances administratives d'une faculté.
- Professeur représente un professeur en exercice dans la faculté étudiée.

2.2.3. Représentation

Les cinq méthodes de bases utilisées pour exploiter les connaissances sont :

- **A** Action
- **CA(c)** Connaissance ajoutée (avec 'c' identifiant le numéro de connaissance)
- **CM(c)** Connaissance modifiée (avec 'c' identifiant le numéro de connaissance)
- **CS(c)** Connaissance supprimée (avec 'c' identifiant le numéro de connaissance)
- **O** Observation

2.2.4. Scénario selon l'entité Administration

Connaissances — Le règlement.

4 août 2001

A : Un bulletin de versement est envoyé à *Etudiant1*.

CA(1): *Etudiant1* est inscrit en première année.

CA(2): *Etudiant1* est redevable d'un minerval.

19 août 2001

CM (2): *Etudiant1* a payé un minerval.

21 août 2001

CA(3): *Etudiant2* est détenteur d'un diplôme étranger.

CA(4): *Etudiant2* doit fournir un document d'équivalence dûment rempli.

CA(5): *Etudiant2* est en cours d'inscription.

A : Un document d'équivalence est envoyé à *Etudiant2*.

24 août 2001

CM(4): *Etudiant2* a fourni un document d'équivalence dûment rempli.

CM(5): *Etudiant2* est inscrit en élève libre en seconde année.

CA(6) : *Etudiant2* est redevable d'un minerval.

A : Un bulletin de virement est envoyé à *Etudiant2*.

29 août 2001

A : Un bulletin de virement est envoyé à *Etudiant3*.

CA(7): *Etudiant3* est inscrit en première année.

CA(8): *Etudiant3* est redevable d'un minerval.

24 septembre 2001

CM(6): *Etudiant2* a payé son minerval.

31 octobre 2001

O : Date du jour : 31 Octobre 2001.

CM (7) Etudiant3 est refusé à l'inscription pour cause de non paiement.

CS (8)

Les connaissances acquises par l'acteur Administration en fin de scénario sont :

Etudiant1 est inscrit et a payé un minerval.

Etudiant2 est détenteur d'un diplôme étranger.

Etudiant2 a fourni un document d'équivalence dûment rempli.

Etudiant2 est inscrit en élève libre et a payé un minerval.

Etudiant3 est refusé à l'inscription pour cause de non paiement.

2.2.5. Scénario selon l'entité Secrétariat

Connaissances	Secrétariat de la faculté Fac. La faculté Fac est composée d'une première et d'une seconde année.
4 août 2001	Aucune information pertinente.
19 août 2001	**CA(1)**: *Etudiant1* est un étudiant inscrit. **CA(2)**: *Etudiant1* est un étudiant de première année.
21 août 2001	Aucune information pertinente.
24 août 2001	Aucune information pertinente.
29 août 2001	Aucune information pertinente.
24 septembre 2001	**CA(3)**: *Etudiant2* est un étudiant inscrit. **CA(4)**: *Etudiant2* est un étudiant de seconde année.
31 octobre 2001	Aucune information pertinente.

Les connaissances acquises par l'acteur Secrétariat en fin de scénario sont :

Etudiant1 est un étudiant légalement inscrit en première année.
Etudiant2 est un étudiant légalement inscrit comme élève libre en seconde année.

2.2.6. Scénario selon l'entité Professeur

Connaissances	Chargé de cours en seconde année de la faculté Fac.
4 août 2001	Aucune information pertinente.
19 août 2001	Aucune information pertinente.
21 août 2001	Aucune information pertinente.
24 août 2001	Aucune information pertinente.
29 août 2001	Aucune information pertinente.
24 septembre 2001	**CA(1)**: *Etudiant2* est un étudiant de seconde année.
31 octobre 2001	Aucune information pertinente.

Les connaissances acquises par l'acteur Professeur en fin de scénario sont :

Etudiant2 est un étudiant de seconde année.

Remarque : La pertinence d'une information en fonction de chaque profil présente une sensibilité différente. Par exemple, un professeur ne doit pas connaître le parcours administratif d'un étudiant, mais seulement une liste des étudiants inscrits à son cours.

2.3. Formulaires : étude de texte à la structure connue

2.3.1. Introduction

Le troisième exemple illustre l'utilisation d'un nouveau support de communication : 'le formulaire'. La particularité d'un formulaire est sa structure parfaitement connue.

De tels documents se composent de deux éléments:

- Un composant textuel
- Un composant dynamique

Le **composant textuel** regroupe des informations sur l'utilisation du formulaire ainsi qu'une liste de questions en ne regroupant que des données intrinsèques à ce formulaire. Toutes les occurrences d'un même type de formulaire ont à tout moment un composant textuel identique.

Le **composant dynamique** regroupe l'ensemble des zones utilisables pour répondre aux questions décrites dans le composant textuel. Ce composant va s'enrichir progressivement lors de la vie du formulaire.

Voici l'extrait d'une fiche interne d'un assureur pour l'acceptation de la couverture médicale d'un assuré. La figure 4 montre une occurrence du formulaire 'Acceptation médicale' à un moment T de sa vie. Le contenu des composants textuels et dynamiques est repris respectivement à la figure 5 et à la figure 6.

Indications de service

N° Relation Société : 10004BC N.° Relation Candidat : 0034 Fiche BR N° Projet : G414

Date Déclench : 01/04/2003 Date envoi pour décision : 12/04/2003

Decision acceptation (Maladie) Service Medical

Accepté normalement le : Accepté conditions spéciales le : 21/04/2003

Conditions : Prime supplémentaire sur risque de 0,5% sur la première tranche du salaire

Ajourné le : A:rev. dans : Refusé le :

Décision Direction HC : Acceptation avec surprime.

figure 4 : Exemple de texte structuré : Acceptation médicale

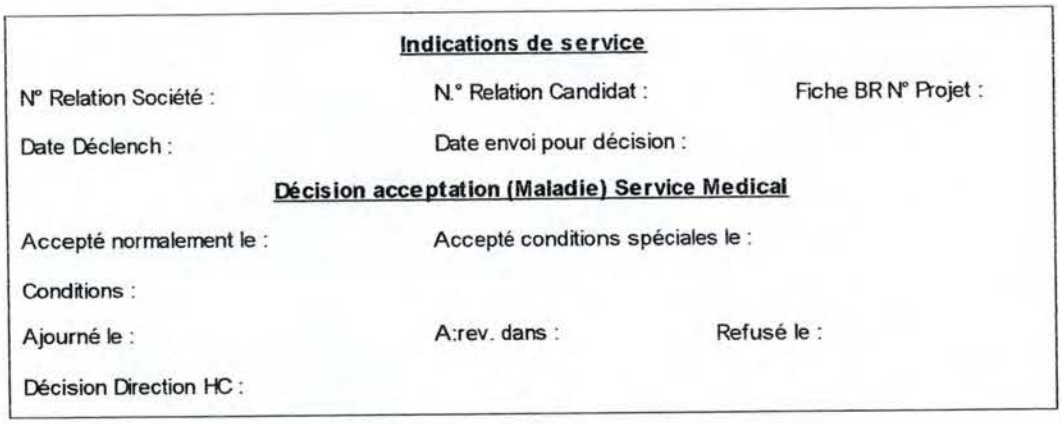


figure 5 : Le composant textuel

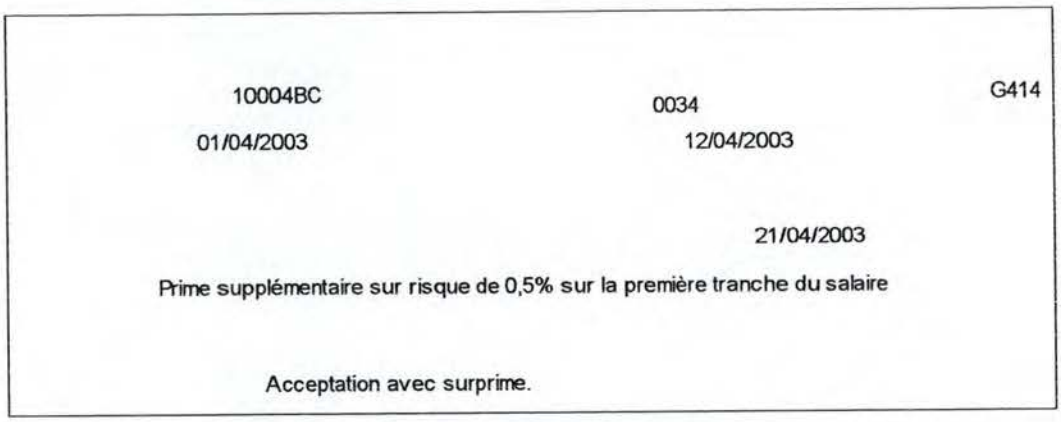


figure 6 : Le composant dynamique

A titre d'exemple remarquons que:

- A la question 'Fiche BR N° Projet' il est répondu 'G414'
- Il n'y a aucune réponse à la question 'Refusé le'

Dans le formulaire 'Acceptation médicale', les réponses se distinguent en deux chapitres logiques. Le premier regroupe les données spécifiques à la gestion interne du dossier ('Indications de service'). Alors que le second bloc reprend toutes les données utiles à la gestion du dossier ('Décision acceptation (Maladie) Service Médical').

2.3.2. Contraintes

Il est possible d'exprimer des contraintes de contenu ou de forme sur les réponses présentes dans le composant dynamique.

Ces contraintes peuvent être de deux types :

- Contrainte de formatage (contr_form).
- Contrainte de dépendance (contr_dep).

Si toutes les contraintes 'contr_form' d'un document sont respectées alors le document est dit **valide**.
Si toutes les contraintes 'contr_form' et 'contr_dep' d'un document sont respectées alors le document est dit **complet**.

cas.formulaire.1

Pour le document 'Acceptation médicale', voici une liste non exhaustive des contraintes et de leur type.

- La réponse à 'Refusé le :' doit être une date valide (contr_form).
- La réponse à 'Décision Direction HC : ' ne doit pas dépasser 80 caractères (contr_form).
- Il faut une réponse pour 'Fiche BR N° Projet' (contr_dep).
- S'il existe une réponse pour 'Accepté condition spéciale le' alors 'Conditions' doit aussi être donné (contr_dep).

2.4. Texte libre : texte à sémantique connue

2.4.1. Introduction

Le dernier exemple étudie le cas d'un document dont une partie de la sémantique est bien connue mais dont la structure pour l'exprimer est inconnue.

Le contexte de l'exemple est l'échange d'informations entre le secrétariat du ministère de la santé publique et un candidat spécialiste en médecine. Un candidat spécialiste en médecine est un étudiant ayant terminé avec succès son premier cycle médical de sept ans et ayant entamé un cycle de spécialisation (chirurgie, radiologie, ...). Le ministère responsable de la reconnaissance des diplômes des candidats spécialistes est le secrétariat de la santé publique.

Pour l'obtention du diplôme de spécialiste, il faut obtenir la reconnaissance de x années, x étant spécifique à chaque spécialité. A chaque fin d'année de spécialisation, le ministère de la santé publique statue sur la réussite de l'année prestée sur base de documents administratifs fournis par le candidat spécialiste.

L'année de spécialisation peut être totalement ou partiellement reconnue. Une année totalement reconnue ajoute une année de reconnaissance au curriculum du candidat spécialiste et une année partiellement reconnue, ce qui est la cas généralement lorsque l'étudiant effectue une année de recherche, ajoute une reconnaissance de moins d'un an au curriculum du candidat spécialiste, p.ex. 6 mois. L'année peut ne pas être reconnue si les résultats sont jugés insuffisants.

2.4.2. Scénario

Voici un exemple de la lettre de reconnaissance envoyée au candidat spécialiste pour notifier l'avis du ministère de la santé. Du fait du caractère confidentiel du document, certaines informations ont été modifiées.

MINISTERE DES AFFAIRES SOCIALES DE LA SANTE PUBLIQUE ET DE L'ENVIRONNEMENT
DIRECTION DE L'ART DE GUERIR

Dr Prescriptus
Avenue des peupliers
Bruxelles

Docteur,

Discipline: SPECIALISTE BOBOLOGIE

Concerne : Introduction de(s) carnet(s) de stage années 4
introduit(s) le 09/10/2002 Réunion du 09/10/2002

La Chambre compétente de la Commission de votre discipline, après avoir examiné les documents repris en rubrique, a donné un avis favorable.

La Commission estime que cette année de stage a été excellente. Cependant elle attire votre attention sur le fait que vous devez compléter votre activité clinique. Votre année de recherche sera comptabilisée pour 6 mois de stage. Vous devez donc introduire un complément de stage de 6 mois. Vous terminez donc la formation en mars 2004.

Le candidat qui ne respectera pas la procédure sera considéré en arrêt de formation et 1'INAMI en sera averti immédiatement avec perte du numéro de candidat spécialiste et toutes les conséquences qui en découleront.

La Commission vous demandera de lui faire parvenir, dès la fin de 1'année de stage, le carnet de stage est visé par le Maître de stage et accompagné des documents repris ci-après:

- le relevé de l'activité de consultation,
- le relevé de l'activité scientifique: congrès, séminaires,...
- une liste récapitulative par prestations et actes techniques

Veuillez agréer, Docteur, l'expression de notre considération distinguée.

figure 7 : Texte libre : lettre de reconnaissance

A la réception du document 'lettre de reconnaissance', le candidat spécialiste se pose la question : *'Quelle est la reconnaissance de mon année ?'*. La réponse à cette question est la seule information pertinente pour le candidat spécialiste.

La réponse à cette question est comprise dans les deux phrases suivantes :

- donnée1 : La Chambre compétente de la Commission de votre discipline a donné un avis favorable.
- donnée2 : Votre année de recherche sera comptabilisée pour 6 mois de stage.

Les informations reprises dans donnée1 peuvent se synthétiser par :

- Qui : l'autorité
- Action: reconnaît
- Quoi : l'année prestée

Nous avons donc localisé la sémantique contenue dans le document. La difficulté provient de la méconnaissance de la structure utilisée pour transmettre celle-ci, dans notre cas la structure utilisée est le langage naturel.

Voici illustrées d'autres structures représentant une sémantique identique, du moins selon le point de vue du candidat spécialiste.

Qui ?	Action	Quoi ?
Le ministre	émet	un avis positif pour 6 mois.
Le secrétaire du ministre	transmet	une reconnaissance d'une demi année.
Le ministère	prononce	une réussite d'un demi cycle.

Pour donner un sens à un texte, et donc être apte à répondre à une question s'y rapportant, il nous faut extraire le contenu sémantique du texte et pouvoir l'interpréter en faisant abstraction de la structure utilisée.

2.5. Acteurs et leurs supports de communication

2.5.1. Introduction

Avant de pouvoir modéliser le monde de la communication et lui offrir différents services, il est indispensable d'en maîtriser son essence. Le fonctionnement de toutes communications est matérialisé par deux éléments : son support et les acteurs y prenant part.

Le support de communication est le média utilisé pour transmettre l'information, les acteurs sont les producteurs et/ou consommateurs de l'information.

2.5.2. Les acteurs

Le monde de la communication met en oeuvre plusieurs types d'acteurs dont une liste non exhaustive est donnée. La nature évolutive du monde de la communication impose une modélisation offrant un maximum de souplesse, plutôt qu'une modélisation complète évitant l'obsolescence du système.

Un acteur reçoit une identification différente en fonction de son domaine de matérialisation. Voici résumées les différentes identifications d'acteurs avec leur contexte générateur :

- Une adresse postale dans le cadre des services postaux.
- Une adresse électronique dans le cadre des messageries électroniques.
- Un numéro téléphonique ou un numéro de fax dans le cadre des services téléphoniques.
- Un nom et prénom dans le cadre des personnes physiques.

2.5.3. Les modes de communication

Les modes de communication peuvent se synthétiser en fonction des acteurs y prenant part.

- Le texte, n'ayant ni expéditeur ni destinataire.
- Le fax, ayant un expéditeur de type numéro téléphonique et un destinataire de type numéro téléphonique.
- La brochure, ayant un destinataire de type adresse postale.
- L'article, ayant un expéditeur de type nom et prénom.
- Le courrier, ayant un destinataire de type adresse postale et peut-être un expéditeur de type adresse postale.
- Le courrier électronique, ayant au moins un expéditeur de type adresse électronique et au moins un destinataire de type adresse électronique.
- L'appel téléphonique, ayant un expéditeur de type appel téléphonique et un ou plusieurs destinataires de type numéro téléphonique.
- La conversation, ayant un expéditeur de type nom et prénom. Il peut exister un ou plusieurs destinataires de type nom, prénom.

En conclusion, le type support de communication utilisé est défini par ses acteurs.

2.5.4. Problème des identifications multiples

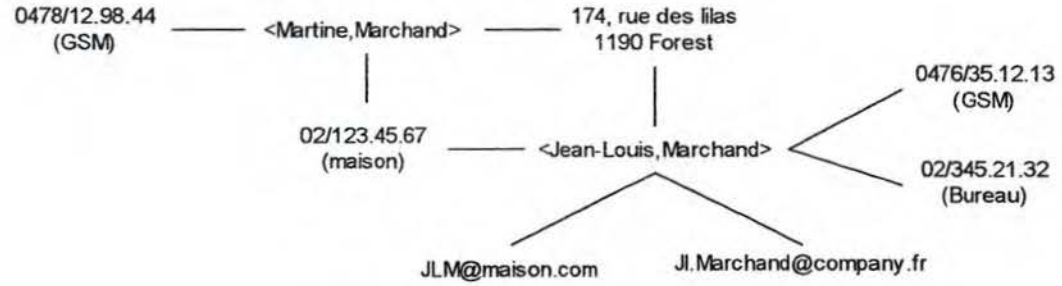


figure 8 : Multiplicité d'acteurs

Le monde de la communication met en scène une identification hétérogène des acteurs. Mais un même acteur peut être identifié par plusieurs catégories d'identification. Un même acteur de la communication peut notamment être identifié par plusieurs adresses électroniques (messagerie professionnelle et personnelle), mais un même numéro de téléphone peut identifier plusieurs acteurs (téléphone familial). Pour détecter les différentes identifications d'un même acteur, une conceptualisation des méthodes de recherche est indispensable.

Voici déjà certaines pistes de réflexion :

- Un courrier permet de lier un couple (nom, prénom) avec une adresse.
- Comment gérer les homonymes ?
- La composition d'une adresse électronique pourrait permettre d'identifier un couple (nom, prénom), p.ex. `jl.marchand@comp.fr` et (`Jean-Louis, Marchand`).
- Utilisation de bases de données externes comme les répertoires téléphoniques (pages d'or).

3. Spécification du monde de la communication

Dans cette partie, les éléments statiques du monde de la communication ayant un impact sur sa capacité sur son évolutivité sont mis en exergue. En effet, la définition et le rôle des acteurs (2.5.3) dans le monde de la communication sont spécifiés et raisonnablement stabilisés. Par contre, les éléments composant la notion 'Acteur' doivent pouvoir évoluer. L'évolution du monde de la communication pourrait exiger la création de nouveaux types d'acteurs, comme une identification GPS (coordonnée planétaire) ou l'identification d'un nœud réseau (dans le cas hypothétique de réseaux intelligents qui prendraient part à une communication).

3.1. Eléments de base

3.1.1. Définitions

Les définitions suivantes installent les fondations de la spécification du monde de la communication.

Soit l'ensemble des matérialisations **Matérialisation** = $\{m_i\}$ avec $i \geq 0$, composé de l'ensemble des caractères utilisables pour représenter un élément.(caractères, chiffres,...).

spec.base.1

Soit un ensemble d'atomes **Atome** = $\{ a_i\}$, avec $\forall i$: $i \geq 0$, a_i étant un mot construit avec les éléments de Matérialisation. L'ensemble Atome est la représentation de tous les mots pouvant être écrits par l'alphabet défini en *spec.base.1*.

spec.base.2

Soit un tuple t, $t = (a_0,..,a_i)$ avec $n \geq i \geq 0$, $a_i \in$ Atome

spec.base.3

La dimension d'un tuple est défini comme : soit $t = (a_1,a_2,a_3,..a_n) \Rightarrow$ dimension(t) =n

p.ex. dimension((un, deux, trois, quatre)) = 4

spec.base.4

Soit TupleI l'ensemble des tuples incomplets, TupleI = { t_i}, avec $n \geq i \geq 0$, $a_i \in$ Atome $\vee$ $a_i = \varnothing$

par exemple:. ti = (a_0,,a_2,..a_n).

spec.base.5

Tout tuple (voir *spec.base.3*) appartient à l'ensemble TupleI.

spec.base.6

Un tuple t est le complémentaire d'un tuple incomplet ti,

si avec t = (a_0,a_1,a_2,..a_n) et ti = (ai_0,ai_1,ai_2,..ai_n) $\Leftrightarrow$ dimension(i) = dimension(ti) et $\forall ai_i \neq \varnothing \Rightarrow ai_i = a_i$.

spec.base.7

Soit un type d'ensemble

Zone = {t_i} avec $\forall$i: #(Zone) $\geq$ i,j ≥ 0 : $t_i \in$ Tuple et dimension(t_i)=dimension(t_j)

spec.base.8

Soit un type d'ensemble

Espace = {$Zone_i$} avec $i \geq 0$,

$\forall$ i,j, $n \geq i,j \geq 0$: $i \neq j \Rightarrow Zone_i \cap Zone_j = \varnothing$

spec.base.9

Soit un type d'ensemble

Espace = {z_i} avec $\forall$i #(Espace) $\geq i \geq 0$: z_i est un ensemble de type Zone

spec.base.10

3.1.2. Exemple : l'espace Codification

Codification est un espace spécialisé regroupant un ensemble de Zones, permettant le stockage de codifications telles que les codes postaux ou les codes pays.

Soit repris ci-dessous un extrait de l'espace Codification :

```
Codification = {(Pays,32,Belgique),(Pays,33, France),(Pays,49,Allemagne),
                (Localité,32,1190,Forest),(Localité,32,1180,Uccle),...}
```

L'ensemble Atome requis pour construire Codification se compose des éléments suivants :

```
Atome = {Pays, Localité, Belgique, France, Allemagne, Forest, Uccle, 32, 33, 49, 1190,
         1180, ... }
```

Dans l'extrait de l'espace Codification sont repris deux Zones :

- de dimension 3 : ('Pays', code Pays, description) codifiant un pays
- de dimension 4 : ('Localité', code Pays, code postal, description) codifiant des localités.

L'extrait de Codification permet de répondre à ces différentes questions :

- Quel est le pays représenté par le code 32 ?
- Quel est le code pays de la localité 'Uccle' ?

3.2. Acteurs et leurs supports

Voici présentée une infrastructure permettant de persister l'ensemble des informations transitant par le monde de la communication en mettant en scène les différents acteurs, les supports, ainsi que l'ensemble des relations liant ceux-ci.

3.2.1. Définition des Acteurs

Soit les ensembles de type Zone (*spec.base.8*)

NonSpécifique = $\{(description)_i\}$
Humain = $\{(nom,prénom)_i\}$
Courrier = $\{(numéro, boîte, rue,localité, pays)_i\}$
Téléphone = $\{(international, préfixe, numéro)_i\}$
Courriel = $\{(nom, serveur, domaine)_i\}$

spec.acteur.1

Soit l'ensemble

Acteur = NonSpécifique ∪ Humain ∪ Courrier ∪ Téléphone ∪ Courriel

spec.acteur.2

L'ensemble Acteur est défini comme un méta-ensemble reprenant toutes les zones spécifiant un type d'Acteur.

3.2.2. Définition des supports

Soit les ensembles de type Zone (*spec.base.8*)

TexteLibre = $\{(texte_n\}$ avec $n \geq 0$: $texte_n \in$ Atome (spec.base.2).

TexteLibre représente tous les textes non structurés pouvant être construits à l'aide d'Atomes.

MessageElectronique = $\{(titre, texte)_n\}$ avec $n \geq 0$: titre, texte $\in$ Atome.

Fax = $\{(en\text{-}tête, texte)_n\}$ avec $n \geq 0$: en-tête, texte $\in$ Atome.

spec.acteur.3

Soit l'ensemble

Contenu = TexteLibre ∪ MessageElectronique ∪ Fax

spec.acteur.4

L'ensemble Contenu est défini comme un méta-ensemble regroupant toutes les zones spécifiant

un type de Contenu.

3.2.3. Définition des relations

Soit l'ensemble de type Zone(*spec.base.8*)

Comparables = { $Comparable_i$ } avec $n \geq i \geq 0$: $Comparable_i \in$ Atome.

avec $\forall c \in$ Comparables, c identifie de manière unique une méthode définie comme

Comparable (x , y) $\rightarrow$ bool avec x et y $\in$ Acteur $\wedge$

$Comparable_i$ (x , y) = $Comparable_i$ (y , x)

Nous dirons que x est synonyme de y lorsque $\exists i \mid Comparable_i$ (x , y)

spec.acteur.5

Soit l'ensemble de type Zone (*spec.base.8*)

Annexes = {$Annexe_i$} avec $n \geq i \geq 0$.: $Annexe_i \in$ Atome.

avec $\forall a \in$ Annexes, a identifie de manière unique une méthode définie comme :

Annexe (x , y) $\rightarrow$ bool avec x et y $\in$ Contenu

Nous dirons que y ne peut être interprété sans le contexte de x lorsque

$\exists i \mid Annexe_i$ (x , y)

spec.acteur.6

Soit l'ensemble **A** de type Annexes, **A** = {a_i} avec $0 \leq i \leq n$

$\exists k$ avec $0 \leq k \leq n \mid a_k$(x , y) => $\forall j$ avec $j \neq k : \neg a_j$(y , x)

spec.acteur.7

Soit l'ensemble de type Zone (*spec.base.8*)

Intervenants = { $Intervenant_i$} avec $n \geq i \geq 0$: $Intervenants_i \in$ Atome.

avec $\forall i \in$ Intervenants, i identifie de manière unique une méthode définie comme

Intervenants (a , c) $\rightarrow$ booléen avec a $\in$ Acteur et c $\in$ Contenu

Nous dirons que a justifie l'existence de c lorsque $\exists i \mid Intervenants_i$ (a , c)

spec.acteur.8

Exprimons la contrainte qu'une annexe ne puisse avoir d'Intervenant :

Soit x et y $\in$ Contenu, $m_j \in$ Acteur, #Acteur $\geq j \geq 0$,

$\exists i \mid \text{Annexe}_i(x,y) \ \forall j,k \ \neg\text{Intervenant}_j(m_k,y)$

spec.acteur.9

Soit l'ensemble:

`RelationsActeur` = Comparables $\cup$ Annexes $\cup$ Intervenants

spec.acteur.10

3.2.4. Exemple: L'ensemble Comparables

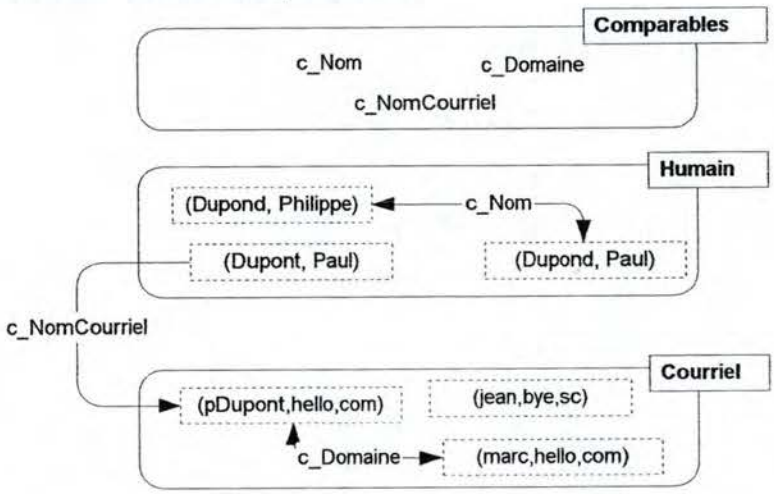


figure 9 : Exemple spécification : 'acteurs et leurs supports'

Dans la figure 9, nous distinguons trois ensembles : Humain, Courriel et Comparables.

Dans l'ensemble Comparable est défini les trois méthodes :

- c_Nom (h_1,h_2) , h1, h2 $\in$ Humain (*spec.acteur.1*) avec h_i = (nom_i,$prénom_i$) et $1 \leq i \leq 2$

  c_Nom (h_1,h_2) $\Leftrightarrow$ nom_1 = nom_2 et 1er caractère $prénom_1$ = 1er caractère $prénom_2$.

- c_Domaine(c_1,c_2), c1,c2 $\in$ Courriel (*spec.acteur.1*) avec c_i = (nom_i,$server_i$,$domaine_i$) et $1 \leq i \leq 2$

  c_Domaine(c_1,c_2) $\Leftrightarrow$ $server_1$=$server_2$ $\wedge$ $domaine_1$=$domaine_2$

- c_NomCourriel(h_1,c_2), $h_1 \in$ Humain et $c_2 \in$ Courriel (*spec.acteur.1*)

  avec h_1 = (nom_1,$prénom_1$) et c_2 = (nom_2,$server_2$,$domaine_2$)

  c_NomCourriel(h_1,c_2) $\Leftrightarrow$ nom_2 = 1er caractère $prénom_1$ + nom_1

3.2.5. Exemple : Les ensembles Annexes et Intervenants

L'analyse des données du message4 défini en 2.1.2 permet l'illustration de l'utilisation des ensembles Intervenants et Annexes.

Les ensembles de type Zone défini dans l'espace sont :

`Courriel` = { (Ami, company, com), (Etud, fundp, be), (Prof, fundp, be) }
`Intervenants` = {Expéditeur, Destinataire }
`Annexes` = { Hyperlien }
`MessageElectronique` = { (Illustration de ..., Regarde ...)}
`TexteLibre` = { http://www.encyclopédie.com }

Pour les relations définies dans Intervenants:

- Expéditeur((Ami, company, com), (Illustration de ..., Regarde ...))
- ¬Expéditeur((Etud, fundp, be), (Illustration de ..., Regarde ...))
- Destinataire((Etud, fundp, be), (Illustration de ..., Regarde ...))

Pour les relations définies dans Annexes:

- Hyperlien(http://www.encyclopedie.com, (Illustration de ..., Regarde ...))
- ¬Hyperlien((Illustration de ..., Regarde ...), http://www.encyclopedie.com)

3.3. Les formulaires

3.3.1. Définitions des composants

Soit l'ensemble de type Zone (*spec.base.8*)

Validations ={ EstUneDate, EstPasVide, EstUnNombre, TailleMax80, TailleMax210 }

avec chaque élément définissant un type de validation.

spec.form.1

Soit les types d'ensembles

Questions = {(Intitulé, ensValidations)$_i$}

étant muni d'une relation d'ordre stricte.

avec ∀i: (Intitulé,Validation)$_i$

Intitulé ∈ Atome ∧ ensValidations ⊂ Validations

Chapitres = {(Intitulé, ensQuestions)$_i$}

étant muni d'une relation d'ordre stricte.

avec ∀i: Intitulé ∈ Atome

et ensQuestions un ensemble de type Questions

Reponses = {(Chapitre, Question, Réponse)$_i$}

avec Chapitre, Question, Réponse ∈ Atome

Formulaire = {ensChapitres$_i$}

avec ∀i:ensChapitres$_i$ un ensemble de type Chapitres.

spec.form.2

Soit l'ensemble de type Zone (*spec.base.8*)

Formulaires = {Intitulé$_i$}

avec ∀i: Intitulé$_i$ identifiant un ensemble de type Formulaire.

spec.form.3

3.3.2. Définition des relations

Soit le type d'ensemble

Formats(F) = { (réponse, validation)$_i$}

avec

F, un ensemble de type Formulaire.

réponse, un élément d'un ensemble de type Questions $\subset$ F.

validation, un élément d'un ensemble de type Validations.

spec.form.4

Soit le type d'ensemble

Contraintes = { (base, implication)$_i$ }

avec

base un ensemble de type Format, base(F1) = { {($r1_1$,$v1_1$),..., ($r1_n$,$v1_n$)} }, n>0

implication, un ensemble de type Format, implication(F2) = { {($r2_1$,$v2_1$),..., ($r2_m$,$v2_m$)} }, m>0

et **F1 = F2**

Soit un élément (b,i) $\in$ **C** un ensemble de type Contraintes, si les validations exprimées dans b sont valides alors les contraintes exprimées dans i doivent aussi être valides. Quand tous les éléments de Contraintes sont valides alors le formulaire est dit complet.

spec.form.5

3.3.2. Exemple : le formulaire 'Acceptation médicale'

Le composant textuel du formulaire exposé en figure 4 peut se décrire selon la structure :

```
Question1 = {  ('N° Relation Societe : ',{EstPasVide,EstUnNombre}),
               ('N° Relation Candidat : ',{EstPasVide,EstUnNombre}),
               ('Fiche BR N° Projet :',{EstPasVide,EstUnNombre}),
               ('Date Déclench : ', {EstUneDate})
               ('Date envoi pour décision : ',{EstUneDate}) }

Question2 = {  ('Accepté normalement le :', {EstUneDate})
               ('Accepté conditions spéciales le :', {EstUneDate})
               ('Conditions', {TailleMax80})
               ('Ajourné le : ', {EstUneDate})
               ('A:rev dans : ', {EstUneDate})
               ('Refusé le : ', {EstUneDate})
               ('Décision Direction HC : ' {TailleMax80}) }

ChapExempl = { ('Indications de service',Question1),
               ('Décision acceptation (Maladie) Service Médical', Question2) }

Formulaires = {ChapExempl} l'ensemble est un singleton dans l'état actuel de nos
                           connaissances

avec ChapitreExempl, un ensemble de type Chapitres
et Question1, Question2 des ensembles de type Questions
```

Le composant dynamique exposé en figure 6 peut se décrire selon la structure :

(Par soucis de concision certains éléments trop longs sont coupés par les caractères '...'.)

```
RepExempl = {  ('Indications de service', 'N° Relation Societe : ', '10004BC' ),
               ('Indications de service', 'N° Relation Candidat : ', '0034'),
               ('Indications de service', 'Fiche BR N° Projet : ', 'G414' ),
               ('Indications de service', 'Date Déclench : ', '01/04/2003'),
               ('Indications de service', 'Date envoi pour décision : ', '12/04/2003'),
               ('Décision acc...', 'Accepté normalement le :', ''),
               ('Décision acc...', 'Accepté conditions spéciales le :', '21/04/2003'),
               ('Décision acc...', 'Conditions', 'Prime supplémentaire sur ris...'),
               ('Décision acc...', 'Ajourné le : ', ''),
               ('Décision acc...', 'A:rev dans : ', ''),
               ('Décision acc...', 'Refusé le : ', ''),
               ('Décision acc...', 'Décision Direction HC : ', 'Acceptation av...') }

Avec RepExempl un ensemble du type Reponses.
```

Les expressions de contraintes plus complexes prennent la forme :

```
C = { ({(('Décision acc...', 'Accepté conditions spéciales le :'), EstPasVide)},
      {(('Décision acc...', 'Conditions'), EstPasVide)}) }
Avec C un ensemble de type Contraintes
```

C exprime le fait que si la date d'une acceptation avec condition est définie alors le champ conditions doit être rempli (pour décrire la condition).

3.4. L'analyseur sémantique

3.4.1. Introduction

Le but de l'analyseur sémantique n'est pas d'arriver à la parfaite compréhension d'un texte libre, mais de conduire à la découverte potentielle d'un contenu spécifié. Son objectif est de positionner une infrastructure permettant l'évolution aisée vers une autre langue de même que l'extension vers des structures grammaticales complexes.

L'analyseur sémantique reçoit une phrase type, représentant la question posée, et un texte représentant le texte à analyser. Au moyen de différentes projections, l'analyseur tente de faire converger la question vers une phrase du texte donné. S'il est possible de trouver une convergence, alors nous dirons que le texte donné contient la sémantique exprimée par la question.

3.4.2. Définitions

Soit les ensembles d'Atomes (*spec.base.2*)

Genre = { masculin, féminin }
Nombre = { singulier, pluriel }
Personne = { première, seconde, troisième }
Temps = { IndicatifPrésent, IndicatifFutur, IndicatifImparfait ,IndicatifPasseSimple,
SubjonctifPresent, Infinitif, SubjonctifImparfait, ParticipePresent,
ImperatifPresent, ConditionnelPrésent, ParticipePasse }

spec.analyseur.1

Soit l'ensemble de type Zone (*spec.base.8*)

Verbe = { (forme, infinitif, temps, nombre, personne)$_i$ }
avec
forme, infinitif ∈ Atome (*spec.base.2*).
temps ∈ Temps, nombre ∈ Nombre, personne ∈ Personne.

forme représente la forme conjuguée de l'élément définie par les éléments : temps, nombre et personne. p.ex.: (aie, avoir, SubjonctifPresent, singulier, première)

spec.analyseur.2

Soit l'ensemble de type Zone (*spec.base.8*)

Nom = { (forme, primitif, nombre, genre)$_i$ }

avec

forme, primitif ∈ Atome (*spec.base.2*).

nombre ∈ Nombre, genre ∈ Genre.

forme représente un nom à la forme '*nombre+genre*'.

primitif représente le nom *nom* à la forme 'masculin+singulier'.

p.ex. : n = (aïeules, aïeul, pluriel, féminin) avec n ∈ Nom.

spec.analyseur.3

Soit l'ensemble de type Zone (*spec.base.8*)

Adjectif = { (forme, primitif, nombre, genre)$_i$ }

avec

forme, primitif ∈ Atome (*spec.base.2*).

nombre ∈ Nombre, genre ∈ Genre.

forme représente un adjectif à la forme '*nombre+genre*'.

primitif représente l'adjectif *adjectif* à la forme 'masculin+singulier'.

p.ex. : a = (adjoints, adjoint, pluriel, masculin) avec a ∈ Adjectif.

spec.analyseur.4

Soit l'ensembles de type Zone (*spec.base.8*)

Adverbe = { (forme)$_i$ } avec i > 0

avec

forme ∈ Atome.

forme reprend la forme de l'adverbe étudié.

spec.analyseur.5

Soit l'ensembles de type Zone (*spec.base.8*)

Article = { (forme,nombre, genre)$_i$ } avec i > 0

avec

forme ∈ Atome (*spec.base.2*).

nombre ∈ Nombre, genre ∈ Genre.

forme reprend la forme de l'article étudié.

spec.analyseur.5

Soit l'ensemble de type Zone (*spec.base.8*)

Définitions = Verbe ∪ Nom ∪ Adjectif ∪ Article ∪ Adverbe

spec.analyseur.6

Soit l'ensemble de type Zone (*spec.base.8*)

Fonctions = { fonction$_i$} avec i > 0

∀i : fonction$_i$ identifie un ensemble de l'ensemble Définitions.

spec.analyseur.7

Dans l'état de nos connaissances, l'ensemble Fonctions regroupe les éléments suivants :

Fonctions = { 'Verbe', 'Nom', 'Adjectif', 'Article', 'Adverbe'}

Soit l'ensemble de type Zone (*spec.base.8*)

Construction = { construction$_n$ } avec ∀ i > 0 : construction$_i$ ∈ Atome

Construction énumère toutes les entités grammaticales disponibles dans l'étude d'un texte.

p.ex.: Construction = {Phrase, sujet, complément d'objet direct, verbe}.

spec.analyseur.8

Soit l'ensemble

Constructions = { (construction, Fonction)$_i$ }

avec

construction ∈ Construction et Fonction un ensemble ordonné de Fonctions.

spec.analyseur.9

Soit l'ensemble de type Zone

Synonymes = { $(mot_0, \{mot_1, mot_2, ..., mot_n\})_i$ } avec $n \geq 1$,

avec

$\forall i : 0 \leq i \leq n : mot_i \in$ Atome.

$\forall i,j : 0 \leq i,j \leq n : i \neq j : mot_i \neq mot_j$.

Nous dirons que les mots mot_i $(1 \leq i \leq n)$ peuvent être remplacés par mot_0 sans perte de sens sémantique.

spec.analyseur.10

Soit l'ensemble ordonné

Mots = { $(mots, fonction)_i$} avec $i > 0$

avec mots $\in$ Atome $\wedge$ fonction $\in$ Fonctions.

spec.analyseur.11

Soit l'ensemble ordonné

Phrase : { $(construction, mots)_i$} avec $i > 0$

avec construction $\in$ Construction et mots $\in$ Mots

spec.analyseur.12

La phrase 'Ceux de qui je me plains, pour qui je travaille' [Grevisse] ,

se représente sous la forme:

{ (phrase, {(ceux, ϕ), (de, ϕ), (qui, ϕ), (je, ϕ), (me, ϕ), (plains, ϕ), (pour, ϕ), (qui, ϕ), (je, ϕ}, (travaille, ϕ)}) }

L'élément (Ceux, ϕ) dénote une fonction (encore) inconnue pour le mot: ceux.

3.4.3. Exemple : Constructions

Voici un liste non exhaustive de **Constructions** dont les exemples sont repris de [Grevisse].

construction	*exemple*
phrase,{ sujet, verbe, adverbe, complément}	Nous travaillons assidûment.
phrase,{ sujet, verbe, adverbe, verbe, complément}	J'ai assidûment travaillé.
phrase, { sujet, verbe adverbe, adjectif, complément}	Il agit très dignement.
phrase, {sujet, verbe}	La terre tourne.
phrase,{sujet, verbe, complément}	Le chien conduit l'aveugle.
phrase,{ sujet, verbe, complément, proposition, phrase}	Je vous avertis **que** vous vendrez le nécessaire **si** vous achetez le superflu.

3.4.4. Exemple : Analyseur sémantique

Les données suivantes reprennent les différents éléments illustrant le fonctionnement de l'analyseur sémantique.

Les ensembles utilisés pour l'analyseur sont:

```
soit S un ensemble de type Synonymes
S = { (autorité, {ministre, secrétaire du ministre, ministère}),
      (émettre, {prononcer, transmettre}) }

soit C un ensemble de type Construction
C = { (sujet, {article, nom}, (phrase, {sujet, verbe, complément}) )

soit N une ensemble de type Nom
N = { (autorité, autorité, singulier, féminin) }

Soit V un ensemble de type Verbe
V = { (émets,    émettre,   IndicatifPrésent, singulier, troisième),
      (prononce, prononcer, IndicatifPrésent, singulier, troisième) }

soit A une ensemble de type Article
A = { (le, singulier, masculin) }
```

Les données

Question: Q = 'Le ministre émet [compléments].'

Le Texte T = '... Le ministère prononce une réussite complète. ...'

La propriété [compléments] exprime le fait que l'élément n'est pas significatif. Ces éléments utilisables dans une question doivent appartenir à l'ensemble Construction (*spec.analyseur.6*).

Le traitements des données

Phase1: Découpe de T en phrases

T = '...' 'Le ministère prononce une réussite complète.' '...'

Phase 2: analyse des phrases

Des transformations sont appliquées aux phrases à l'aide des ensembles N, V, A et C:

- L'ensemble V permet d'identifier le mot 'émets' comme étant un verbe.
- La construction dans C : `(phrase, {sujet, verbe, complément})` localise le sujet.
- La construction dans C : `((sujet, {article, nom})` détermine les éléments du sujet.

Les différentes transformations s'écrivent sous la forme :

Q = le ministre [verbe=émet] [complément]

Q = [sujet=le ministre] [verbe=émet] [compléments]

Q = [article=le] [nom=ministre] [verbe=émet] [compléments]

T = '...'

'[article=le] [nom=ministère] [verbe=prononce] [complément= une réussite complète]'

'...'

Phase 3: Projection par synonyme

Les verbes sont remplacés par leur infinitif.

Tous les éléments sont projetés sur leur synonyme à l'aide de l'ensemble S (si existant).

Les phrases s'écrivent sous la forme:

Q = [article=le] [nom= autorité] [verbe=émettre] [compléments]

T = '...'

'[article=le] [nom= autorité] [verbe=émettre] [complément=une réussite complète]'

'...'

Phase 4: Convergence.

Une convergence existe si tous les éléments définis dans Q sont retrouvés dans une phrase de T.

3.5. Le scénario dynamique

3.5.1. Définitions

Soit le type d'ensemble

`Observations` = { o_i } avec i > 0.

Tout o ∈ Observations est défini comme une opération sur un Espace e produisant impérativement un élément d'un ensemble de type Zone contenu dans un espace e. En fin d'opération, l'espace e peut être altéré.

spec.scenario.1

Soit le type d'ensemble

`Traitements` = { t_i } avec i > 0.

Tout t ∈ Traitements est défini comme une succession d'opérations se produisant sur un ou plusieurs espaces. Ces différents espaces peuvent être altérés en fin d'opération.

spec.scenario.2

Soit l'ensemble

`SynchronisationOT` = { $(O, T)_i$ } avec i > 0

avec

O, un ensemble de type Observations.

T, un ensemble de type Traitements.

Lorsque les différentes observations sont accomplies, les traitements peuvent s'exécuter à l'aide des différentes informations fournies par les observations.

spec.scenario.3

Soit l'ensemble

SynchronisationTO = { $(T, O)_i$ } avec $i > 0$

avec

O, un ensemble de type Observations

T, un ensemble de type Traitements

Lorsque les différents traitements sont exécutés, toutes les observations peuvent se déclencher.

spec.scenario.4

Soit l'ensemble

Extractions = {$extraction_i$} avec $i \geq 0$, $extraction_i \in$ Atome.

L'ensemble Extractions représente les points de connexion vers le monde extérieur qui permettent à un Espace d'exporter une connaissance vers le monde extérieur. Chaque extraction fourni une méthode 'envoi(z:Zone, ext:Extension) définissant son mode de communication.

p.ex: Extractions = {*Courriels, Fax, SMS* }

spec.scenario.6

3.5.2. Exemple : Première phase d'inscription

Introduction

Le scénario proposé en 2.2.2. illustre les différents cheminements administratifs pour l'inscription d'un étudiant. Le dynamise est illustré à partir de ce fragment du scénario :

- Un courrier est reçu contenant la demande d'inscription en première année de *Etudiant1*. La demande d'inscription est complète et valide.

Le scénario dynamique peut s'écrire sous la forme (voir les notations en Annexe2):

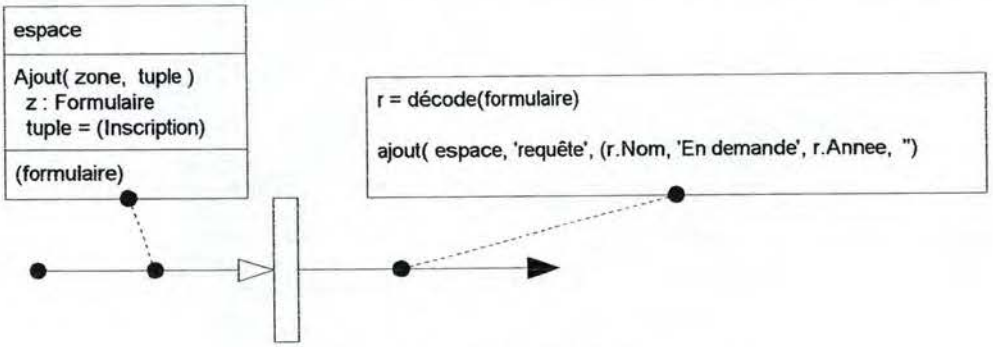


figure 10 : Exemple de scénario dynamique

Définition de l'ensemble de type Espace : 'espace'

Le formulaire de demande d'inscription est défini comme suit :

Quest = { ('Nom:', {EstPasVide}), ('Année d'inscription:', {EstPasVide}) }
avec Quest, un ensemble de type Questions (*spec.form.2*).

Chap = { 'Formulaire de demande d'inscription', Quest }
avec Chap, un ensemble de type Chapitres (*pec.form.2*).

Rép = { ('Formulaire de demande d'inscription', 'Nom:', ''),
('Formulaire de demande d'inscription', 'Année d'inscription:', ''),}
avec Rép, un ensemble de type Réponses.

Inscription = { Chap }
avec Inscription, un ensemble de type Formulaire (*spec.form.2*).

Le type de Contenu défini sur l'Espace espace est :

Requête = { (nom, année, état, texte)$_i$ }

Contenu = { Requête } (*spec.acteur.4*)

Cheminement du scénario dynamique

La réception du courrier : " Formulaire de demande d'inscription Nom: Jacques Sanzot Année d'inscription: 1ere informatique" provoque un traitement manuel et l'ajout d'un formulaire *Inscription* dans espace.

L'ajout de la demande d'inscription dans l'espace satisfait l'observation initiale de la figure 10. Le formulaire retrouvé est mis à disposition pour tous les traitements définis dans le point de synchronisation.

Les traitements sur l'espace sont :

1. Décoder le formulaire (*décode(formulaire)*) pour obtenir les réponses (*r*).

2. Ajouter une requête dans espace avec les données de la demande d'inscription (*ajout(espace...)*.

4. Spécification du monde de la communication

La définition de tous les composants de données dynamiques (connaissances) décrivant le monde de la communication à été faite au chapitre 3. Ce chapitre spécifie toutes les méthodes utilisables pour gérer les connaissances du monde de la communication.

remarque:
Le préfixe {ob} est utilisé pour préciser que la méthode appartient à l'ensemble Observations (spec.scenario.1).

Le préfixe {tr} est utilisé pour préciser que la méthode appartient à l'ensemble Traitements (spec.scenario.2).

Un élément défini avec une première lettre majuscule est à considérer comme un ensemble.
Un élément défini avec une première lettre minuscule est à considérer comme un élément d'un ensemble.

- Elément : Ensemble → Elément est un ensemble de type Ensemble
- élément : Ensemble → élément ∈ Ensemble

4.1. Spécification de 'tuple'

Sorte:

générique : Atome ∪ élément vide

position : entier

Définition:

Soit t un tuple avec $t = (e_1, e_2, .., e_n)$ avec $\forall i,\ 1 \leq i \leq n : e_i \in$ générique.

Spécification:

{tr} **tuple_Créer** (e_1 : générique, e_2 : générique, .., e_i : générique) → t : tuple

Pré	$i > 0$
Post	$t = (e_1, e_1, .. , e_n)$

{tr} **tuple_Dimension** (t : tuple) → entier

Pré	$t = (e_1, e_2, .., e_n) \wedge n > 0$
Post	n

{tr} **tuple_Complet** (t : tuple) → booléen

Pré	$t = (e_1, e_2, .., e_n) \wedge n > 0$
Post	vrai $\Leftrightarrow \forall i: 1 \leq i \leq n : e_i \neq$ vide

{tr} **tuple_Synonyme** (t : tuple, ts : tuple) → booléen

Pré	$t = (e_1, e_2, .., e_n) \wedge ts = (e'_1, e'_2, .., e'_n) \wedge$ tuple_Complet(t)
Post	vrai $\Leftrightarrow \forall i : 1 \leq i \leq n : e'_i \neq$vide $\Rightarrow e'_i = e_i$

{tr} **tuple_Composant** (t : tuple, p : position) → e : générique

Pré	$t = (e_1, e_2, .., e_n) \wedge 1 \leq p \leq n$
Post	$e = e_p$

4.2. Spécification du type d'ensemble Zone

Sorte:

description : chaîne de caractères

Définition:

Soit Z un ensemble de type Zone avec $Z = \{t_1, t_2, .., t_n\}$ avec $\forall i : 0 < i \leq n$: tuple_Complet(t_i)

Spécification:

{tr} **zone_Créer** (dimension : entier) → Z : Zone

Pré	dimension > 0
Post	Z = {}

{tr} **zone_Dimension** (Z : Zone) → entier

Pré	$Z = \{ t_1, t_2, .., t_n \}$
Post	zone_Dimension(zone_Créer (x)) = x

{tr} **zone_Ajout** (Z: Zone, t : tuple)

Pré	$Z = \{t_1, t_2, .., t_n\} \wedge$ tuple_Dimension(t) = zone_Dimension(Z)
Post	zone_existe(t) $\Rightarrow Z' = Z$ $\neg$zone_existe(t) $\Rightarrow Z' = Z \cup \{t\}$

{tr} **zone_Existe**(Z: Zone, t : tuple) → booléen

Pré	$Z = \{t_1, t_2, .., t_n\}$
Post	zone_Ajout (Z, t) ⇒ zone_Existe (Z, t)

{tr} **zone_Edite**(Z : Zone, ts : tuple) → t : tuple

Pré	$Z = \{t_1, t_2, .., t_n\} \wedge ts = (e_1, e_2, .., e_n)$
Post	$Z' = Z$ $t = z \Leftarrow \exists z \in Z \mid$ tuple_Synonyme(z, ts) $\wedge$ zone_Existe(Z, z) $\vee$ t= ()

{tr} **zone_Supprime**(Z : Zone, ts : tuple) → t : tuple

Pré	$Z = \{ t_1, t_2, .., t_n\} \wedge ts = (e_1, e_2, .., e_n)$
Post	tuple_Synonyme(t, ts) $\wedge$ zone_Existe(Z, t) $\wedge Z' = Z \setminus \{t\}$

4.2. Spécification du type d'ensemble Espace

Sorte:

Définition:

Soit E un ensemble de type Espace avec $E=\{z_1, z_2, .., z_n\}$ avec $\forall i : 0 < i \leq n : z_i \in$ Zone

Spécification:

{tr} **espace_Créer** → E : Espace

Pré	
Post	E = {}

{tr} **espace_Ajout**(E : Espace, Z : Zone) → Espace

Pré	$E=\{z_1, z_2, .., z_n\} \wedge Z = \{ t_1, t_2, .., t_n\}$
Post	$E' = E \cup Z$

{tr} **espace_Existe_Zone**(E : Espace, Z : Zone) → booléen

Pré	$E=\{z_1, z_2, .., z_n\} \wedge Z = \{ t_1, t_2, .., t_n\}$
Post	espace_Existe_Zone(espace_Ajout_Zone(E, Z))

{tr} **espace_Zone**(E : Espace, Z : Zone) → Zone

Pré	$E=\{z_1, z_2, .., z_n\} \wedge Z = \{ t_1, t_2, .., t_n\}$
Post	espace_Zone(espace_Ajout(E, Z)) = Z

4.3. Spécification des relations

Sorte:

Définition:

Soit R un ensemble de type Zone avec $R = \{t_1, t_2, .., t_n\}$ avec $\forall i, 0 < i \leq n$: tuple_Complet(t_i)

L'ensemble des relations est un ensemble respectant les contraintes des ensembles de type Zone mais supportant les primitives supplémentaires définie dans spécification.

Spécification:

{tr} **zone_Ajout** (Z : Zone, a : Atome, Z1 : Zone, Z2 : Zone)

[réécriture de la primitive définie dans Zone]

Pré	$Z = \{t_1, t_2, .., t_n\}$, $Z1 = \{t1_1, t1_2, .., t1_n\}$, $Z2 = \{t2_1, t2_2, .., t2_n\}$
Post	zone_existe(a) $\Rightarrow Z' = Z$ $\neg$zone_existe(a) $\Rightarrow Z' = Z \cup \{a\}$ Il existe une méthode 'Validation(tuple, tuple) $\rightarrow$ booléen' identifié de manière unique par a.

{tr} **relation_Validation_Lien** (Z : Zone, a : Atome, t1 : tuple, t2 : tuple) $\rightarrow$ bool

Pré	$Z = \{t_1, t_2, .., t_n\}$ tuple_Complet(t1) $\wedge$ tuple_Complet(t2) relation_Ajout(Z, a, Z1, Z2) $\Rightarrow$ t1 $\in$ Z1 $\wedge$ t2 $\in$ Z2
Post	vrai : si la fonction de validation **identifiée par a** est satisfaite pour les contraintes Validation(t1, t2).

{tr} **relation_Ajout_Lien** (Z : Zone, a : Atome, t1 : tuple, t2 : tuple)

Pré	$Z = \{t_1, t_2, .., t_n\}$ $\wedge$ tuple_Complet(t1) $\wedge$ tuple_Complet(t2)
Post	relation_Validation_Lien(Z, a, t1, t2)

{tr} **relation_lien_source** (Z : Zone, a : Atome, ts1 : tuple, ts2 : tuple) $\rightarrow$ Ens(Tuple)

Pré	$Z = \{t_1, t_2, .., t_n\}$
Post	tx, ty $\in$ Tuple tx $\in$ Result $\Leftrightarrow$ tuple_Complet(tx) $\wedge$ tuple_Complet(ty) $\wedge$ tuple_Synonyme(tx, ts1) $\wedge$ tuple_Synonyme(ty, ts2) $\wedge$ relation_Validation_Lien(Z, a, tx, ty)

{tr} **relation_lien_destination** (Z : Zone, a : Atome, ts1 : tuple, ts2 : tuple) → Ens(Tuple)

Pré	$Z = \{t_1, t_2, .., t_n\}$
Post	tx, ty ∈ Tuple ty ∈ Result ⇔ tuple_Complet(tx) ∧ tuple_Complet(ty) ∧ tuple_Synonyme(tx, ts1) ∧ tuple_Synonyme(ty, ts2) ∧ relation_Validation_Lien(Z, a, tx, ty)

4.4. Les formulaires

4.4.1. Spécification de l'ensemble strictement ordonné

Sortes:

t : paramètre

position : entier

Ens(t) : ensemble strictement ordonné d'éléments de type t

Définition:

Soit EO = $(e_1, e_2, .. , e_i, .., e_n)$ avec $e_1 < e_2 < e_i < e_n$ et $n \geq 0$. Si n=0, alors EO ={}.

Spécification:

{} **ensemble_Créer:** → EO : Ens(t)

Pré	
Post	EO = {}

{} **ensemble_Appartient**(EO : Ens(t), e : t) → bool : booléen

Pré	$EO = (e_1, e_2, .., e_n)$
Post	$\neg bool \Rightarrow (n=0) \vee (\forall i, 1 \leq i \leq n, e_i \neq e)$

{} **ensemble_Ajouter** (EO: Ens(t), p : position, e : t)

Pré	$EO = (e_1, e_2, .., e_n) \wedge p > 0$
Post	$EO = \{\} \Rightarrow E' = \{e\}$ $EO = \{e_1,..., e_{p-1}, e_p, e_{p+1},..., e_n\} \Rightarrow E' = \{e_1, .., e_{p-1}, \mathbf{e}, e_p, e_{p+1},..., e_n\}$ $EO = \{e_1,...,e_n\} \wedge p>n \Rightarrow E' = \{e_1,..., e_n, \mathbf{e}\}$

{} **ensemble_Enlever**(EO : Ens(t), p : position)

Pré	$EO = (e_1, e_2, .., e_n)$
Post	$EO = \{\} \Rightarrow EO' = \{\}$ $EO = \{e_1,..., e_{p-1}, e_p, e_{p+1},..., e_n\} \Rightarrow EO' = \{e_1, .., e_{p-1}, e_{p+1},..., e_n\}$ $EO = \{e_1,..., e_n\} \wedge p>n \Rightarrow EO' = \{e_1,..., e_n,\}$

{} **ensemble_Editer**(EO : Ens(t), p : position) → e : t

Pré	$EO = (e_1, e_2, .., e_n) \wedge p>0$
Post	$EO = \{\} \Rightarrow e = vide$ $EO = \{e_1,..., e_{p-1}, e_p, e_{p+1},..., e_n\} \Rightarrow e = e_p$ $EO = \{e_1,..., e_n\} \wedge p>n \Rightarrow e = vide$

{} **ensemble_Position**(EO : Ens(t), e : t) -> p : position

Pré	$EO = (e_1, e_2, .., e_n)$
Post	$EO = \{\} \Rightarrow p = 0$ $EO = \{e_1,..., e_{i-1}, e, e_{i+1},..., e_n\} \Rightarrow p = i$ $EO = \{e_1,..., e_{i-1}, e_i, e_{i+1},..., e_n\} \wedge \forall j, 1 \leq j \leq n: ej \neq e \Rightarrow p = 0$

{} **ensemble_Premier**(EO : Ens(t)) -> e : t

Pré	$EO = (e_1, e_2, .., e_n)$
Post	e = ensemble_Editer(EO, 1)

{} **ensemble_Suivant**(EO : Ens(t), e : t) -> e_2 : t

Pré	$EO = (e_1, e_2, .., e_n) \wedge$ ensemble_Position(EO, e) ≥ 1
Post	e_2 = ensemble_Editer (EO, ensemble_Position(EO, e) + 1)

{} **ensemble_Précédent**(EO : Ens(t), e : t) -> e_2 : t

Pré	$EO = (e_1, e_2, .., e_n) \wedge$ ensemble_Position(EO, e) $\leq n$
Post	e_2 = ensemble_Editer (EO, ensemble_Position(EO, e) - 1)

4.4.2. Spécification des types d'ensembles : Questions et Chapitres

Sortes:

intitulé : Chaîne de caractères

question : Chaîne de caractères

chapitre : Chaîne de caractères

réponse : Chaîne de caractères

validation : fonction de prototype validation():booléen

Spécification:

{} **question_Créer**() → EO : Ens(validation)

Pré	/
Post	EO = {}

{} **chapitre_Créer**( ) → EO : Ens(question)

Pré	/
Post	EO = {}

{} **formulaire_Créer( ) → EO : Ens(chapitre)**

Pré	/
Post	EO = {}

4.4.3. Spécification de l'ensemble Formulaires

Sorte:

Questions, Chapitres, Réponses (voir *spec.form.3)*

Formulaires: voir *spec.form.3*

Contraintes : voir *spec.form.5*

Spécification:

{tr} Fonction **formulaire_Décodage**(F:Formulaire, T: chaînes de caractères) → R:Réponses

Décodage d'une chaîne de caractère t à partir des contraintes définies dans le formulaire F.

remarques: Par soucis de concision, nous ne représenterons pas la partie ensValidations des éléments de l'ensemble Questions.

Pré	$F = \{ (c_1, ensQ_1), ..., (c_i, ensQ_i), ..., (c_n, ensQ_n) \}$ avec n>0 Soit les ensembles $ensQ_i$ ($1 \leq i \leq n$) définis comme des ensembles de type Questions $\forall i : 1 \leq i \leq n : ensQ_i = \{(q_{n(i-1)+1},\{\}),...,(q_{n(i)},\{\})\}$ avec $n(0) = 0$ $n(1) = \#ensQ_1$, $n(i) = n(i-1)+\#ensQ_i$ L'énoncé d'une question se représente sous la forme d'une liste de caractères $\forall k : 1 \leq k \leq n(n) : q_k = (c_{1,k}, ..., c_{taille(k),k})$ et $taille(k) = \#q_k$ avec $\forall p : 0 \leq p \leq taille(i) : c_{p,i} \in$ Matérialisation. $T = (t_1, t_2, ..., t_p)$ avec $p > 0$ et $\forall i: t_i \in$ Matérialisation.

Post

cas 1 : R={}

Le texte donné T n'a pu être intégré dans les contraintes du formulaire.

cas 2 : R≠{}

$R = \{ (c_1, q_1, r_1), ..., (c_1, q_{t(1)}, r_{t(1)}), ..., (c_1, q_{n(1)}, r_{n(1)}),$

$(c_2, q_{n(1)+1}, r_{n(1)+1}), ..., (c_2, q_{t(2)}, r_{t(2)}), ..., (c_2, q_{n(2)}, r_{n(2)}), ...,$

$(c_i, q_{n(i)+1}, r_{n(i)+1}), ..., (c_i, q_{t(i)}, r_{t(i)}), ..., (c_i, q_{n(i)}, r_{n(i)}),$

$..., (c_n, q_{n(n)}, r_{n(n)}) \}$

avec

$\forall i: 1 \leq z \leq n: \; n(z-1)+1 \leq t(z) \leq n(z)$

Soit $(c_i, q_{t(i)}, r_{t(i)}), r_{t(i)}$ la réponse à la $t(i)^{ème}$ question du $i^{ème}$ chapitre.

$$\forall i,k: 2 \leq i \leq n \wedge 1 \leq k(i) \leq taille(i): \; c_{k(i),i} = t_z; z = k(i) + \sum_{j=1}^{i-1}(offset(j) + taille(j))$$

$$\text{et } offset(n) = p - (taille(n) + \sum_{j=1}^{n-1}(offset(j) + taille(j)))$$

DécodageFormulaire.1

$$\forall i: 1 \leq i \leq n: r_i = t_z; taille(i) + \sum_{j=1}^{i-1}(offset(j) + taille(j)) + 1 \leq z \leq \sum_{j=1}^{i}(offset(j) + taille(j))$$

DécodageFormulaire.2

Exemple:

Soit T = 'abcdefghijkmlno'

Soit le formulaire F = {(c_1, ensQuestions$_1$), (c_2, ensQuestions$_2$) }

ensQuestions$_1$ = {(q_1,{}), (q_2,{}), (q_3,{}) }

q_1='ab'

$c_{1,1}$ = 'a'

$c_{2,1}$='b'

q_2='def'

$c_{1,2}$ = 'd'

$c_{2,2}$='e',

$c_{3,2}$ = 'f'

q_3='gh'

$c_{1,3}$ = 'g'

$c_{2,2}$='h'

ensQuestions$_2$ = {(q_4,{}) }

q_4='l'

$c_{1,4}$ = 'l'

Sous forme graphique, voici représenté le formulaire :

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
a	**b**	c	**d**	**e**	**f**	**g**	**h**	i	j	k	**l**	m	n	o

Les caractères du composant statique sont placés en gras.

Les données utiles au calculs sont :

- taille(1)=2
- taille(2)=3
- taille(3)=2;
- taille(4)=1
- n=4 et p=15

En appliquant la formule *DécodageFormulaire.1*:

(i=2) $1 \leq k(2) \leq 3 : c_{k(2),2} = t_{k(2) + taille(1) + offset(1)}$

En exprimant les différentes valeurs pour k(i), nous pouvons écrire:

- $c_{1,2} = t_{3+offset(1)} = 'd'$
- $c_{2,2} = t_{4+offset(1)} = 'e'$
- $c_{3,2} = t_{5+offset(1)} = 'f'$

Nous concluons que offset(1) = 1

(i=3) $1 \leq k(3) \leq 2 : c_{k(3),3} = t_{k(3) + taille(1) + offset(1) + offset(2) + taille(2) = t_{k(3) + 6 + offset(2)}}$

En exprimant les différentes valeurs pour k(i), nous pouvons écrire:

- $c_{1,3} = t_{7+offset(1)} = 'g'$
- $c_{2,3} = t_{8+offset(1)} = 'h'$

Nous concluons que offset(2) = 0

(i=4) de manière identique → offset(3) = 3

Et offset(4) = 15 - (1 + (2+1) + (3+0) + (2+3)) = 3

En appliquant la formule *DécodageFormulaire.2*:

- $r_1 = [t_3;t_3] = 'c'$
- $r_2 = [t_{3+(2+1)+1};t_{(2+1)+(0+3)}] = [t_{(7)};t_{(6)}] = ''$
- $r_3 = [t_{2+(2+1)+(0+3)+1};t_{(2+1)+(0+3)+(2+3)}] = [t_{(9)};t_{(11)}] = 'ijk'$
- $r_4 = [t_{(13)};t_{(15)}] = 'mno'$

{tr} **formulaire_validation**(fct : Validations , réponse : Réponse)

Pré	
Post	vrai : réponse répond aux exigences de la validation de type fct.

{tr} **formulaire_EstValide**(F : Formulaire, R : Réponses) : booléen

Pré	$F = \{ (c_1, ensQ_1), ..., (c_i, ensQ_i), ..., (c_n, ensQ_n) \}$ avec $n>0$ $ensQ_1 = \{(q_1,ensV_1), ...,(q_{n(1)}, ensV_{n(1)})\}$ avec $n(1) = \#ensQ_1$ $ensQ_i = \{(q_{n(i-1)+1}, ensV_{n(i-1)+1}),...,(q_{n(i)}, ensV_{n(i)})\}$ avec $n(i) = n(i-1)+\#ensQ_i$ $ensQ_n = \{(q_{n(n-1)+1}, ensV_{n(n-1)+1}),...,(q_{n(n)}, ensV_{n(n)})\}$ avec $n(n)=n(n-1)+\#ensQ_n$ avec $\forall i : 1\leq i \leq n : ensQ_i \subset$ Questions $\forall i : 1\leq i \leq n(n) : ensValid_i \subset$ Validations $R = \{ (c_1, q_1, r_1), ..., (c_1, q_{t(1)}, r_{t(1)}), ..., (c_1, q_{n(1)}, r_{n(1)}),$ $(c_2, q_{n(1)+1}, r_{n(1)+1}), ..., (c_2, q_{t(2)}, r_{t(2)}), ..., (c_2, q_{n(2)}, r_{n(2)}), ...,$ $(c_i, q_{n(i)+1}, r_{n(i-1)+1}), ..., (c_i, q_{t(i)}, r_{t(i)}), ..., (c_i, q_{n(i)}, r_{n(i)}),$ $..., (c_n, q_{n(n)}, r_{n(n)}) \}$ avec: $\forall i: 1 \leq z \leq n: n(z-1)+1 \leq t(z) \leq n(z)$
Post	vrai $\forall t : 1 \leq t \leq n(n) : \forall fct \in ensV_t :$ formulaire_validation(fct,r_t)

{tr} **formulaire_EstComplet**(R : Réponses, C : Contraintes) : booléen

Pré	$R = \{ (c_1, q_1, r_1), \ldots, (c_1, q_{t(1)}, r_{t(1)}), \ldots, (c_1, q_{n(1)}, r_{n(1)}),$ $(c_2, q_{n(1)+1}, r_{n(1)+1}), \ldots, (c_2, q_{t(2)}, r_{t(2)}), \ldots, (c_2, q_{n(2)}, r_{n(2)}), \ldots,$ $(c_i, q_{n(i)+1}, r_{n(i-1)+1}), \ldots, (c_i, q_{t(i)}, r_{t(i)}), \ldots, (c_i, q_{n(i)}, r_{n(i)}),$ $\ldots, (c_n, q_{n(n)}, r_{n(n)}) \}$ avec $\forall i: 1 \leq z \leq n: \; n(z-1)+1 \leq t(z) \leq n(z)$ $C = \{ (\{(rb,vb)_k\}, \{(ri,vi)_n\})_p \}$ avec $k,n,p > 0$ et rb,ri ∈ Réponses et vb,vi ∈ Validations
Post	vrai ⇐ $\forall c \in C \mid c=(\{(rb,vb)_k\}, \{(ri,vi)_n\}) \Rightarrow$ (($\forall k$: **formulaire_validation**(vb,rb)) ⇒ (($\forall n$: **formulaire_validation**(vi,ri))

{tr} **analyseur_Proj_Infinitif**(Demande : Phrase, V : Verbe) → Phrase

Projection de tous les verbes de l'ensemble Phrase donné sur leur forme infinitive.

Pré	Demande = { (phrase, { (m_1, f_1), (m_2, f_2), ..., (m_n, f_n)})} avec n>0 V = {(forme, infinitif, temps, nombre, personne)$_m$} avec m>0
Post	v ∈ Atome, t ∈ Temps, n ∈ Nombre, p ∈ Personne ∀i : 1 ≤ i ≤ n : $f'_i = f_i$ $m'_i = m_i \Leftarrow f_i \neq$ 'Verbe' $m'_i = m_i \Leftarrow () =$ zone_Edite(V, (c_i, , , ,)) m'i = valeur ⇐ f_i = 'Verbe' ∧ (m_i, v, t, n, p) = zone_Edite(V, (m_i, , , ,))

{tr} **analyseur_Proj_Synonyme**(Demande : Phrase, Syn : Synonymes) → Phrase

Projection de tous les mots de Demande sur leur synonyme (si disponible).

Pré	Demande = { (phrase, { (m_1, f_1), (m_2, f_2), ..., (m_n, f_n)})} avec n>0 Syn = { (synonyme, (mot_1, mot_2, ..., mot_m))$_p$ } avec m,p >0
Post	∀i : 1 ≤ i ≤ n : $f'_i = f_i$ m'i = valeur ⇐ ∃ j>0 ∃syn ∈ Syn | syn= { (valeur, { mot_1, mot_2, ..., mot_m})} ∧ $mot_j = m_i$

{tr} **analyseur_ProjConst_Acceptable**(Demande : Phrase,

const : Constructions) → booléen

Si vrai, alors il est possible d'appliquer la transformation de construction *const* à la phrase *Demande*.

Pré	Demande = { ($fonction_i$, $Mots_i$)$_i$} avec n≥i>0 avec $Mots_i$ = { (m_1, f_1), (m_2, f_2), ..., ($m_{n(i)}$, $f_{n(i)}$) } avec n(i) > 0 const = (construction, {def_1, ..., def_m}) avec m>0 et ∀i: 1 ≤ i ≤ m : def_i ∈ Def

Post	vrai $\Leftarrow \exists p: 0 < p \leq n \mid$ construction = $fonction_p$ $\wedge$ $\exists i : 0<i\leq n_{(p)}\ \exists k : 0<k\leq m \mid def_k = f_i$ (1) $\wedge$ $\forall i,j,k,l: 0<i<j\leq n(p),\ 0 < k,l \leq m \mid def_k = f_i \wedge def_l = f_j \wedge f_i \neq f_j \wedge \Rightarrow f_i < f_j$ (2) $\wedge$ $\forall k : 0<k\leq m\ \square i : 0<i\leq n_{(p)} \mid def_k = f_i \Rightarrow k=m \vee (\exists t : 0<t\leq n_{(p)} \mid def_{k+1}=f_t)$ (3) (1) exprime la contrainte qu'il doit y avoir *au moins* un élément commun entre les éléments de l'ensemble $\{def_1, ..., def_m\}$ et les éléments de l'ensemble $Mots_i$. (2) exprime la contrainte que si deux éléments apparaissent dans les ensembles $\{def_1, ..., def_m\}$ et $Mots_i$ alors leurs ordres doivent être identiques dans les deux ensembles. (3) exprime le fait qu'il n'est pas acceptable d'avoir deux composants fonctions successifs def_i et def_j qui ne soit pas définis dans $Mots_i$.

Exemple :

Demande = { (phrase, { (le, article), (ministre, nom), (émet, verbe), (un, article), (avis,nom)})}

const1 = (phrase, {sujet, verbe, complément})

const2 = (complément, {....})

const3 = (phrase, {verbe, nom, complément})

const3 = (phrase, {sujet, verbe, adverbe, adjectif})

analyseur_ProjConst_Acceptable(Demande, const1)

¬analyseur_ProjConst_Acceptable(Demande, const2)

(1) ⇒ Il n'existe pas un $fonction_i$ = complément

¬analyseur_ProjConst_Acceptable(Demande, const3)

(2) ⇒ dans demande nom < verbe et dans const3 verbe < nom

¬analyseur_ProjConst_Acceptable(Demande, const4)

(3) ⇒ adverbe n'est pas défini dans Demande avec k=3 (k<m)

et il n'existe pas d'adverbe dans Demande.

{tr} **analyseur_ProjConst**(Demande : Phrase, const : Constructions) → Projection : Phrase

Modification de la structure de Phrase de telle manière que la succession des fonctions de Phrase reflète la structure fournie par const.

Pré	Demande = { (fct_1, $mots_1$), ..., (fct_i, $mots_i$), ..., (fct_n, $mots_n$) } avec n>0 avec $Mots_i$ = { (m_1, f_1), (m_2, f_2), ..., ($m_{n(i)}$, $f_{n(i)}$) } avec n(i) > 0 const = (construction, {def_1, ..., def_m}) avec m>0 et $\forall i: 1 \leq i \leq m : def_i \in Def$ analyseur_ProjConst_Acceptable(Demande, const) ∧ construction = fct_i
Post	Demande' = { (fct_1, $mots_1$), ... , (fct_{i-1}, $mots_{i-1}$), (fct_{p+1}, $mots_{p+1}$), ..., (fct_{p+m}, $mots_{p+m}$), (fct_{i+1}, $mots_{i+1}$) ..., (fct_n, $mots_n$) } **avec** $(fct_{i-1}, mots_{i-1}) < (fct_{p+1}, mots_{p+1}) < ... < (fct_{p+m}, mots_{p+m}) < (fct_{i+1}, mots_{i+1})$ $\forall k: 1 \leq k \leq m : fct_{p+k} = def_k$ $\forall k: 1 \leq k \leq m \ \forall l: 1 \leq l \leq n(i) \mid def_k = fct_l \Rightarrow mots_{p+k} = mots_l$ $\forall k: 1 \leq k \leq m \ \square l: 1 \leq l \leq n(i) \mid def_k = fct_l \Rightarrow mots_{p+k} = Mots_i$ avec $k<m \Rightarrow mots_{p+k} = mots_{p+k} \setminus \{ e \mid e \geq mots_{p+k+1}\}$ et $k>1 \Rightarrow mots_{p+k} = mots_{p+k} \setminus \{ e \mid e \leq mots_{p+k-1}\}$

Exemple :

Demande={(phrase,{('le', article), ('ministre', nom), ('émet', verbe), ('un', article), ('avis', nom)})}

const = (phrase, {sujet, verbe, complément})

Projection = analyseur_ProjConst(Demande, const, Def)

Projection = { (sujet, { ('le', article), ('ministre', nom) }),
(verbe, { ('émet', verbe) }),
(complément, { ('un', article), ('avis', nom) }) }

{tr} **analyseur_Proj_Mots**(mots : Mots) → liste_mots : chaîne de caractères

Reconstruction d'un morceau de phrase à partir d'un ensemble Mots

Pré	mots = { $(c_1, f_1), (c_2, f_2), ..., (c_n, f_n)$ } avec n > 0
Post	liste_mots = $c_1 + ' ' + c_2 + ' ' + ... + ' ' + c_n$

{tr} **analyseur_Proj_Phrase**(Demande : Phrase) → phrase : chaîne de caractères

Reconstruction d'une phrase au sens du langage naturel à partir d'un ensemble Phrase

Pré	Demande = { $(fonction_i, mots_i)$} avec i>0 avec $mots_i$ = { $(c_1, f_1), (c_2, f_2), ..., (c_{n(i)}, f_{n(i)})$ } avec n(i) > 0
Post	phrase=[$\forall j : 1 \leq j < i$: (analyseur_Proj_Mots($mots_j$) + ' ')]+ analyseur_Proj_Mots($mots_i$)

4.6. Gestion du scénario dynamique

Les différentes phases précédentes ont permis de spécifier les composants du monde de la communication. Dans l'état actuel du système, nous ne disposons d'aucun outil pour nous permettre une gestion automatisée du flux d'informations transitant par notre système. Nous allons donc offrir les outils pour modéliser un comportement réagissant à un état spécifique de notre système.

{ob} Fonction **scn_Ajout** (E : Espace, Z : Zone, ti : TupleI) → to: tuple

Pré	E = { $z_1, z_2, ..., z_n$ } avec n ≥ 0 Z = { $t_1, t_2, ..., t_m$ } avec m ≥ 0 espace_Existe_Zone(E, Z)
Post	tuple_Synonyme(to, ti) ∧ zone_Existe(Z, to)

{ob} Fonction **scn_Suppr** (E : Espace, Z : Zone, ti:TupleI) : tuple

Pré	E = { $z_1, z_2, ..., z_n$ } avec n ≥ 0 Z = { $t_1, t_2, ..., t_m$ } avec m ≥ 0 espace_Existe_Zone(E, Z)
Post	tuple_Synonyme(to, ti) ∧ zone_Existe(Z, to) ∧ Z' = Z \ { ti }

{tr} Fonction **scn_Envoi** (t : Tuple, ext : Extractions) : tuple

Pré	t = $(e_1, e_2, ..., e_n)$ avec n ≥ 0
Post	t est transmis à ext.

4.7. Exemples

4.7.1. Le dialogue électronique

Dans le chapitre 2.1, nous introduisions un exemple de dialogue électronique. Voici le même scénario repris sous forme des primitives définies dans ce chapitre.

```
E : Espace

Courriel : Zone
c1, c2, c3 : tuple

MessageElectronique : Zone
m1, m2, m3, m4 : tuple

Intervenants : Zone

------ Définition des ensembles utilisables ------
Acteur = zone_Créer(3)

MessageElectronique = zone_Créer(2)

Intervenants = zone_Créer(1)
zone_Ajout(Intervenants, 'Expéditeurs', Acteur, MessageElectronique)
zone_Ajout(Intervenants, 'Destinataires', Acteur, MessageElectronique)

------ Définition du message 1 ------
c1 = tuple_Créer('Etud', 'Universite', 'be')
zone_Ajout(Acteur, c1)

c2 = tuple_Créer('Prof', 'Universite', 'be')
zone_Ajout(Acteur, c1)

m1 = tuple_Créer('besoin d'aide', 'Qu'est-ce une ontologie ?')
zone_Ajout(MessageElectronique, m1)

Relation_Ajout_Lien(Intervenants, 'Expéditeurs', c1)
Relation_Ajout_Lien(Intervenants, 'Destinataires', c2)

------ Définitions du message 2 ------
c2 = tuple_Créer('Prof', 'Universite', 'be')
zone_Ajout(Acteur, c1)

c1 = tuple_Créer('Etud', 'Universite', 'be')
zone_Ajout(Acteur, c1)

m2 = tuple_Créer('Re: besoin d'aide', Simple analytique ...')
zone_Ajout(MessageElectronique, m2)

Relation_Ajout_Lien(Intervenants, 'Expéditeurs', c2)
Relation_Ajout_Lien(Intervenants, 'Destinataires', c1)

------ Construction de manière identique pour message 3 & 4....

----- Définition de l'espace -----
E = Espace_Créer()
Espace_Ajout(E, Courriel)
Espace_Ajout(E, MessageElectronique)
Espace_Ajout(E, Intervenants)
```

5. Application - Automatisation du service de support

5.1. Contexte

Une entreprise désire automatiser sa première ligne de support d'aide aux utilisateurs. Une première ligne de support ne répondant qu'à des questions à faible valeur ajoutée, les questions plus complexes sont alors transmise au niveau le plus élevé représenté par la seconde ligne de support. La finalité est la minimalisation de requêtes transmissent à la seconde ligne de support.

Les trois tâches principales du service de support sont:

1. L'aide aux utilisateurs sur le fonctionnement du matériel de l'entreprise, et ceci compris les différentes applications informatiques.

2. Le traitement des demandes d'installations des différentes applications de l'entreprise.

3. La gestion des droits d'accès aux différentes données disponibles sur le réseau de l'entreprise (base de données, répertoires partagés).

Ces différentes requêtes sont transmise au moyen d'un document type, lequel contient un espace de texte libre pour préciser les raisons de la requête.

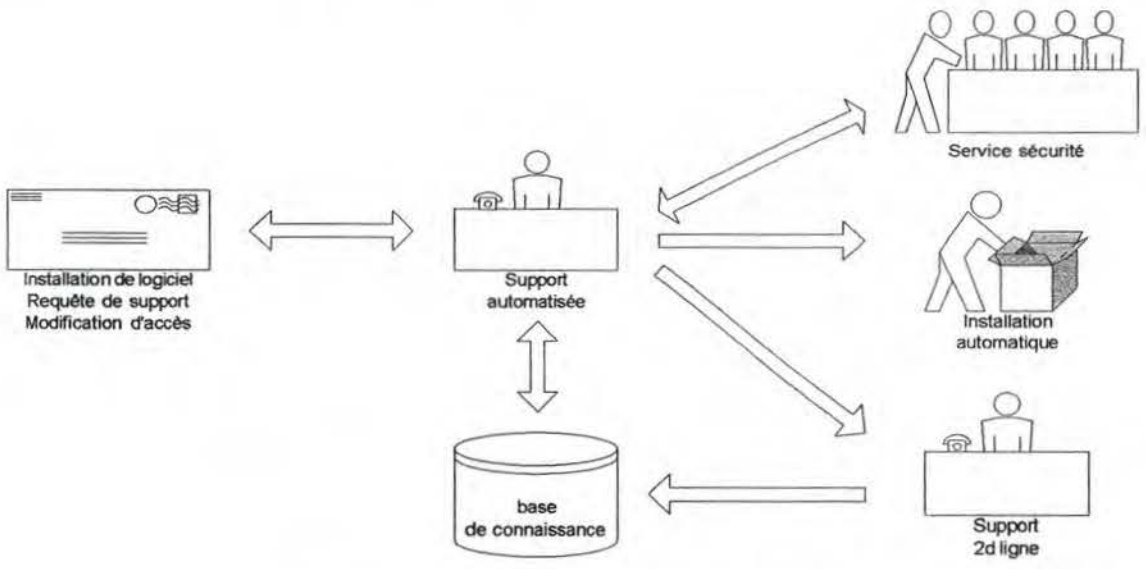


figure 11 : Schéma général du service de support.

5.2. Les procédures

De manière générale, toutes les demandes ne correspondant pas au format de document de support seront renvoyées à l'expéditeur.

Pour les demandes d'informations, le support dispose d'une base de connaissance lui permettant de répondre à un maximum de questions. Si aucune réponse satisfaisante n'est trouvée, la question est transmise à la seconde ligne de support. La résolution du problème sera incluse dans la base de connaissance de la première ligne de support sous forme de fiches d'informations.

Pour toute demande d'installation de logiciel validée par le support de première ligne, une demande d'installation est envoyée au logiciel d'installation automatisée.

Pour les demandes de modification d'accès, après validation par le service de sécurité, le traitement demandé est effectué avec accusé de réception au demandeur en fin d'opération.

De manière générale, il est demandé de conserver toutes demandes d'interventions (pour facturation et historique).

5.3. Le système

Le système d'intervention de première ligne est basé sur un espace 'Support' comprenant toutes les demandes d'interventions techniques. L'espace Support conservera aussi toutes les fiches d'informations sur les problèmes et sur les logiciels.

5.3.1. L'espace Support en détail

L'espace Support est composé des éléments suivants :

- **Acteur = Courriel ∪ PosteDeTravail**
- **Comparable = { Interne }**

  avec Interne(x,y) : x ∈ Courriel et y ∈ PosteDeTravail
- **Intervenants = { Requête }**

  avec Requête(x,y) : x ∈ Courriel et y ∈ Contenu
- **Extractions = {SecondeLigne, Installation, Securité, CourrielInterne }**
- **Contenu = InterventionLogiciel ∪ InterventionAccès ∪ FicheTechnique**
- **Formulaire = { fIntervention }**

La nature de chacun des composants est explicitée ci-dessous lors de son utilisation.

5.3.2. Les Acteurs

Les acteurs du système

Les deux types d'acteurs coexistant dans l'espace Support sont :

- Un adresse électronique interne.

- Un identifiant de poste de travail lié à un utilisateur. Dans l'entreprise étudiée, un poste de travail est assigné spécifiquement à un employé. Un poste de travail dispose de sa propre identification tout comme chaque employé dispose d'une identification réseau.

Les deux ensembles formant l'ensemble Acteurs sont :

Courriel voir *spec.acteur.1*

PosteDeTravail = { (utilisateur, poste)$_i$ }

avec $i \geq 0$ et $\forall i$: utilisateur, poste $\in$ Atome

Les relations

Dans l'entreprise, un employé reçoit une identification réseau et une adresse électronique interne. La relation ***Interne*** permet de lier les deux identifications.

Les Extractions

Les extractions prévues dans le système sont :

- La seconde ligne de support (*SecondeLigne*).

- L'application d'installation automatisée de logiciel (*Installation*).

- Le service de sécurité (*Sécurité*).

- Un acteur interne à l'entreprise (*CourrielInterne*).

5.3.3. Demande d'intervention technique

Introduction:

Le formulaire : 'demande d'intervention' doit être utilisé pour chaque nouvelle demande d'intervention. Le formulaire est décrit en figure 12, les éléments '[*item*]' définissant le composant dynamique (*voir 2.3.*).

Le formulaire de demande d'intervention se compose de trois chapitres :

- Les données de l'utilisateur.
- Une demande d'installation de logiciel.
- Une demande de création ou de modification d'accès. L'élément accès spécifie le type d'accès demandé : lecture, écriture, ...

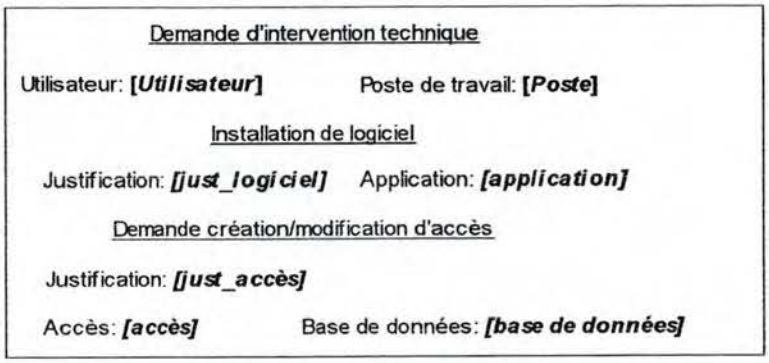


figure 12 : Le formulaire de demande d'intervention

Les contraintes du formulaire :

EstPasVide(Utilisateur)

EstPasVide(Poste)

EstPasVide(just_logiciel) ⇒ EstPasVide(application)

¬EstPasVide(just_logiciel) ⇒ ¬EstPasVide(application)

EstPasVide(just_accès) ⇒ EstPasVide(accès) ∧ EstPasVide(base de données)

¬EstPasVide(just_accès) ⇒ ¬EstPasVide(accès) ∧ ¬EstPasVide(base de données)

Structure du formulaire:

La structure du formulaire peut s'écrire sous la forme (voir 3.3.1.) :

Question1 = { ('Utilisateur: ', {EstPasVide}), ('Poste de travail: ', {EstPasVide}) }

Question2 = { ('Justification: ', {}), ('Application: ', {}) }

Question3 = { ('Justification: ', {}), ('Accès: ', {}), ('Base de données: ', {} }

Chapitres = { ('Demande d'intervention technique', Question1),
('Installation de logiciel', Question2),
('Demande création/modification d'accès', Question3) }

Les alias suivants sont utilisés pour faciliter l'écriture de l'ensemble **Contraintes:**

e_1 = ('Installation de logiciel', 'Justification: ')

e_2 = ('Installation de logiciel', 'Application:')

e_3 = ('Demande création/ modification d'accès', 'Justification')

e_4 = ('Demande création/ modification d'accès', 'Accès')

e_5 = ('Demande création/ modification d'accès', 'Base de données')

Contraintes = { ({(e_1, EstPasVide)}, {(e_2, EstPasVide)}),
({(e_1, EstVide)}, {(e_2, EstVide)}),
({(e_3, EstPasVide)}, { (e_4, EstPasVide), (e_5, EstPasVide}),
({(e_3,EstVide)}, { (e_4, EstVide), (e_5, EstVide}) }

Contenu 'intervention technique':

Pour chaque demande d'information technique, un Contenu de type *InterventionLogiciel* ou *InterventionAccès* est le support de communication pendant le traitement de la demande jusqu'à sa conclusion ou la notification de son refus.

InterventionLogiciel={ (utilisateur, poste, justification, logiciel, état, remarque)$_i$}
avec $i \geq 0$ et utilisateur, poste, justification, logiciel, état, remarque $\in$ Atome

remarque: *état* spécifie l'état de la demande (encours, acceptée, refusée, clôturée, ...)

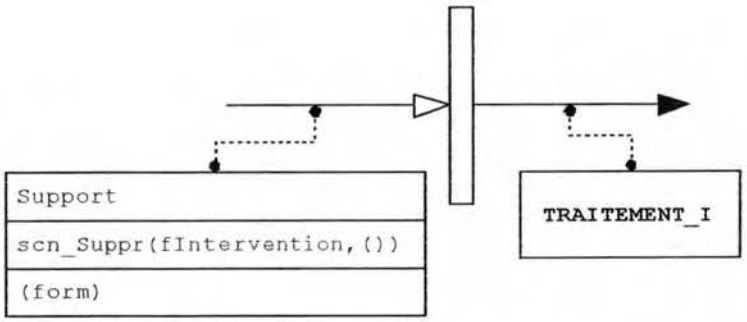
InterventionAccès={(utilisateur, poste, justification, accès, basededonnées, état, remarque)$_i$}
avec $i \geq 0$ et utilisateur, poste, justification, accès, basededonnées, état, remarque $\in$ Atome

remarque: *état* spécifie l'état de la demande (en cours, acceptée, refusée, clôturée, ...)

5.3.4. Scénario dynamique : 'intervention technique'

Voir la description des notations en *Annexe 2*.

Etape 1 : Injection de la demande d'intervention technique:



TRAITEMENT_1

```
tl,ta : tuple
if (form.just_Logiciel <> "")
  tl=tuple_Créer(form.utilisateur, form.poste, form.just_Logiciel, form.application,
                 'valide', '')
  scn_Envoi(tl, Installation)
end if

if (form.just_accès <> "")
  ta=tuple_Créer(form.utilisateur, form.poste, form.just_Acces, form.accès,
                 form.base_de_données, 'valide','')
  scn_Envoi(ta, Sécurité)
end if
```

Les différents états possibles pour une demande d'intervention technique sont repris dans le graphe d'état de la figure 13.

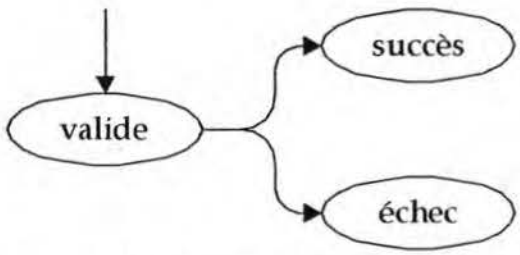


figure 13 : Graphe d'état d'une fiche technique

Etape 2 : Traitement par l'acteur externe *'Installation'*

En réaction à la réception d'un tuple 'InterventionLogiciel', l'acteur externe *'Installation'* va installer le logiciel demandé et effectuer les opérations suivantes:

```
soit t  = (justificatif,application,'succès', remarque)
     tu = (utilisateur, poste)
```

t et tu sont construit à partir des données reçues de la demande d'installation, seul l'élément remarque peut être modifié.

```
zone_Ajout(InterventionLogiciel, t))
relation_Ajout_Lien (Intervenants, Expediteur, tu, t)
```

En cas d'échec d'installation, le même scénario est exécuté à la différence que

- `t=(justificatif, application,'échec', remarque)`
- *remarque* contient les raisons de l'échec.

Le traitement effectué par l'acteur externe 'Sécurité est équivalent.

5.3.5. Fiche technique

Une fiche technique contient la description d'un problème et sa résolution.

FicheTechnique = { (application, description, résolution)$_i$}

avec $i \geq 0$ et application, description, résolution $\in$ Atome

- *application* : logiciel avec lequel s'est produit le problème.
- *description*: description du problème. Une description vide indique que la fiche technique représente la FAQ de *l'application*. Dans ce cas, *résolution* contient le contenu de la FAQ.
- *résolution*: description des opérations à effectuer pour résoudre le problème.

5.3.6. Demande d'information

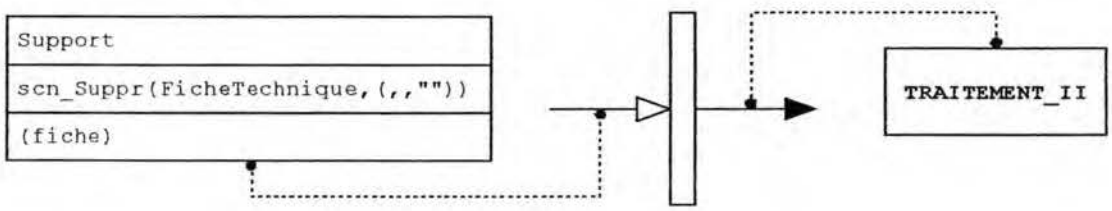
Introduction

Une demande d'information se fait au moyen d'une fiche technique. Les données à fournir sont le nom de l'application (*application*) et une description du problème rencontré (*description*).

Le support de première ligne qui reçoit une demande d'information va rechercher dans les fiches techniques disponibles celle qui répond au mieux au problème rencontré. Lorsque le problème n'est documenté dans aucune fiche technique, le support de première ligne envoi la FAQ spécifique de l'application (champs *description* vide dans la fiche technique). Si aucune FAQ n'existe pour l'application, alors le problème est envoyé directement à la seconde ligne de support qui se charge alors de fournir une réponse à l'utilisateur.

Si la réponse fournie par la première ligne de support n'apporte pas une solution au problème de l'utilisateur alors il lui est possible de contacter directement le support de seconde ligne.

Requête de fiche technique



Description du ***TRAITEMENT_I***

```
ficheTech     : tuple
p             : Phrase
ensCourriel   : Ens(Tuple)
courriel      : Courriel
Dictionnaire  : Définitions
V             : Verbe ' V ∈ Dictionnaire
S             : Synonyme
requête       : chaine de caractères

'----- Analsyse de la requête fournie -----
p = analyseur_Phrase_Créer(fiche.titre)
p = analyseur_Structuration_Créer(p,Dictionnaire)
p = analyseur_Infinitif(p, V)
p = analyseur_Proj_Synonyme(p, S)

requête = analyseur_Proj_Phrase(p)

'----- 1. Recherche de la fiche technique ------
if zone_Existe(FicheTechnique, (fiche.application, requête, ))
  ficheTech = zone_Edite(FicheTechnique, (fiche.application, requête,))
  ensCourriel = relation_lien_source ( Intervenants, Expediteur, (,,), fiche)
  courriel = Ensemble_Premier(ensCourriel)
  scn_Envoi(ficheTech,courriel)

'----- 2. Recherche de la FAQ -----
else if zone_Existe(FicheTechnique, (fiche.application,,))
  ficheTech = zone_Edite(FicheTechnique, (fiche.application,,)
  ensCourriel = relation_lien_source ( Intervenants, Expediteur, (,,), fiche)
  courriel = Ensemble_Premier(ensCourriel)
  scn_Envoi(ficheTech,courriel)

'----- 3. Envoi de la requête vers la seconde ligne de support -----
else
  scn_Envoi(fiche, SecondeLigne)
end if
```

6. Conclusions

La première ambition de ce mémoire est la formation de professionnels aux nouvelles technologies de partage de l'information que sont Linda et JavaSpaces. Une étape importante est déjà atteinte par la rédaction d'un kit de formation et par la présentation de celui-ci à un panel hétérogène d'acteurs du monde de l'informatique. L'utilisation future du kit de formation est déjà planifiée pour un futur proche. L'intégration dans un catalogue de formations impose encore quelques modifications du kit de formation. Les premières exploitations commerciales sont prévues pour le début de l'année 2004 !

La deuxième ambition de ce mémoire est d'offrir une boîte à outils pour le traitement de l'information électronique. Les caractéristiques essentielles devant être sa capacité d'évolution et d'extension. Ces caractéristiques sont atteintes grâce à l'utilisation de notations ensemblistes. Leurs véracités ont été démontrées par un exemple final ajoutant de nouvelles notions aux systèmes comme une fiche technique.

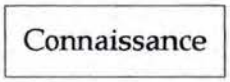
L'analyseur sémantique est l'élément le plus complexe dont les spécifications n'ont été écrites que pour établir les idées de base. Les extensions futures peuvent prendre les directions suivantes:

- spécification de nouvelle projection
- validation du modèle avec l'utilisation d'une autre langue
- la spécification d'une politique de projection sur les structures grammaticales
- politique de convergences

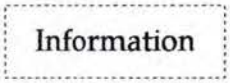
La structure actuelle est déjà construite pour faciliter l'extension vers les nouvelles voies citées.

Annexes

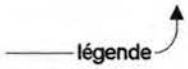
Annexe 1. : Représentation de connaissances



: Connaissance existante dans le système.



: Information : fraction de connaissance.



: Lien sémantique entre deux éléments. Le type de lien est représenté par légende'.

Annexe 2. : Notations pour le scénario dynamique

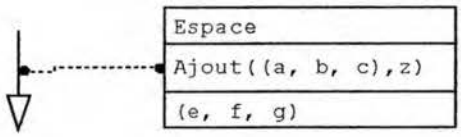
L'observation initiale
(spec.scenario.1)

L'observation initiale est le point de départ d'un scénario dynamique . Il est activé dès que les contraintes de l'observation sont satisfaites. Autrement dit, aucun composant du scénario dynamique n'est actif avant la satisfaction de l'observation initiale. Chaque schéma doit contenir une et une seul observation initiale.

L'observation simple
(spec.scenario.1)

L'initialisation d'un scénario active toutes les observations simples n'ayant aucun prédécesseurs.

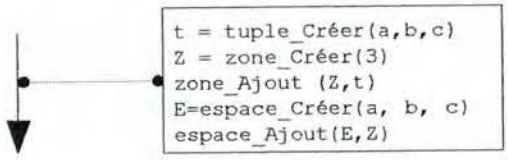
Descripteur d'observation



Les contraintes d'une observation (initiale ou simple) sont définies dans un descripteur d'observation. Dans un descripteur est défini l'espace sur lequel s'applique l'observation, le nom de l'observation (avec ses paramètres) et l'identification du tuple produit (une observation produit toujours un tuple).

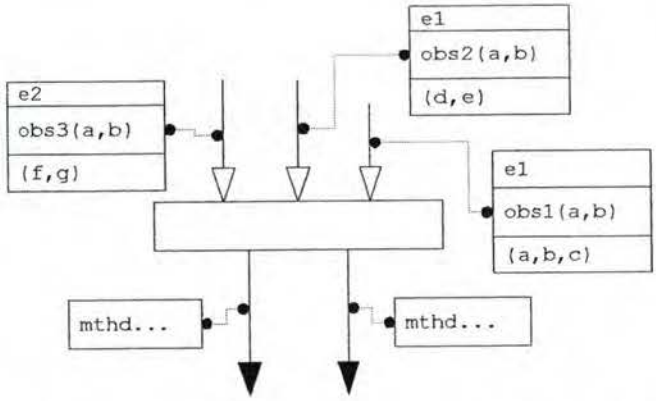
Dans notre exemple, l'observation s'applique à l'espace Espace. Sa méthode est ajout() dont le premier paramètre est le tuple (a,b,c) et le second paramètre est la valeur z. Le tuple retourné par l'observation est (e, f, g).

Traitement
(spec.scenario.2)



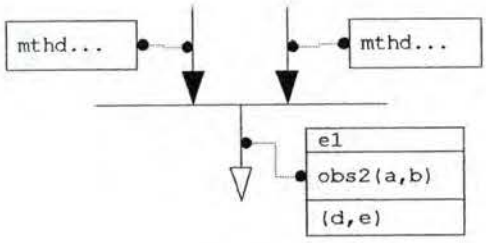
Le descripteur contient l'ensemble des opérations à effectuer par le traitement.

SynchronisationOT
(spec.scenario.5)



Un point de synchronisation OT est composé d'une liste non nulle d'observations (partie supérieure) et d'une liste non nulle de traitements. La satisfaction de toutes les observations autorise l'exécution de tous les traitements. Le résultat d'une observation peut être utilisé par un traitement.

SynchronisationTO
(spec.scenario.6)



Un point de synchronisation OT est composé d'une liste non nulle de traitements (partie supérieure) et d'une liste non nulle d'observations (partie inférieure). Les observations sont activées dès que tous les traitements ont fini leur exécution.

Annexe 3 : Parallélisme, Linda et JavaSpaces

Voici présenté le kit de formation aux technologies suivantes:

- Programmation parallèle
- Philosophie Linda
- JavaSpaces

Ce kit de formation n'a pas comme ambition la formation aux différentes technologies citées, mais l'éveil d'un intérêt auprès de professionnels de l'informatique sur l'utilisation de telles technologies.

Le kit de formation reprend un ensemble de diapositives de présentation. Le texte intégral de la formation n'étant pas fourni, ce kit s'adresse à un formateur averti.

Pour augmenter la valeur pédagogique du kit, des exemples utilisant la technologie JavaSpaces sont fournis.

Programmation parallèle
Linda et Javaspaces

Philippe Verdeyen

Introduction

Comment résoudre un problème complexe demandant une grande puissance de calcul ?

- Division du problème en morceaux moins complexes ?
- Exécution de morceaux de résolutions de manière parallèle ?
- Distribution de la résolution au travers de machines reliées par un réseau ?

Comment **spécifier** ces différentes solutions ?

Comment coordonner les différents processus (asynchrone) ?

Le parallélisme est un moyen prouvé pour une exécution rapide.

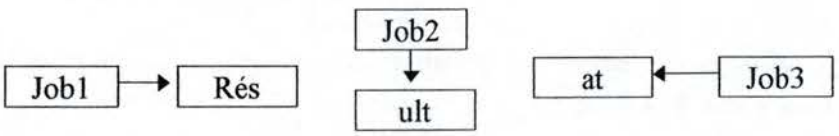
Partie I : Théorie de la programmation parallèle

- Ordonnancement de processus
- Communication inter-processus

Ordonnancement de processus

- Parallélisme par résultat

  La résolution du problème se compose en résultats calculés de manière indépendante. Chaque processus est spécialisé dans le calcul d'un résultat.



Méthodologie:

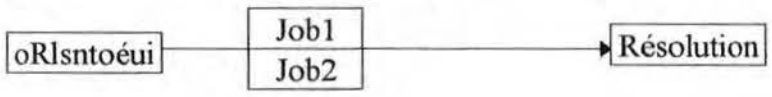
Pour chaque résultat:

1. Définir une structure d'accueil pour le résultat.
2. Lier le processus et sa structure d'accueil.
3. Identifier les besoins de données (input) pour le processus.
4. Définir l'algorithme du processus.
5. Définir un point de terminaison.

Ordonnancement de processus

■Parallélisme par agenda

Une structure de données doit subir un ensemble de transformations pour obtenir le résultat final. Chaque processus est doté de la même compétence. L'augmentation du nombre de processus 'devrait' augmenter la performance du système.



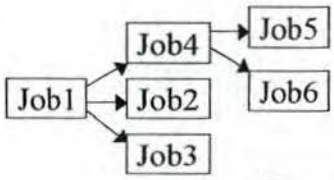
Méthodologie:

1. Définir l'algorithme commun.
2. Définir un mécanisme de coordination entre les processus.
3. Identifier **la granularité** des processus.

Ordonnancement de processus

■Parallélisme par spécialisation

La résolution du problème se représente sous la forme d'un réseau de dépendances entre des tâches indépendantes.



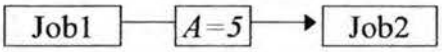
Composants:

- Filtre: conversion de flux de données.
- Client: chargé d'une tâche.
- Serveur: fournisseur d'un service.
- Agent: Client et Serveur.

Distribution des données

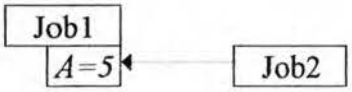
- Communication par message

  Les processus communiquent par l'envoi d'une structure de données. Il n'existe aucune donnée commune. (specialist parallelism).



- Communication par données partagées

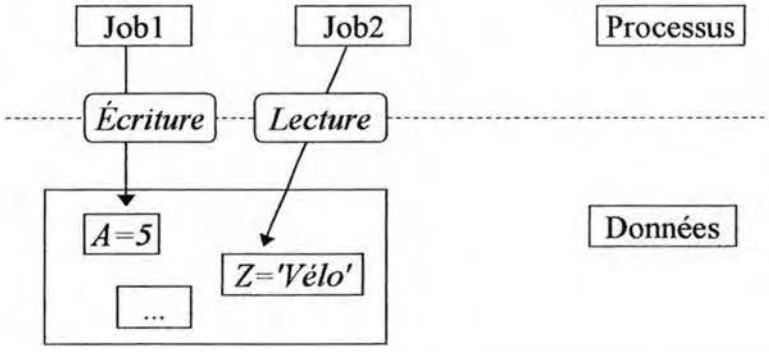
  Chaque processus dispose d'une structure publique de données. Un processus à la possibilité de lire le résultat du calcul d'un autre processus.



Distribution des données

- Communication par données distribuées

  Les données sont regroupées dans une zone publique où les processus peuvent écrire de nouvelles données ou lire des données existantes. Une telle structure doit offrir des services d'accès coopérants.



Modélisation par approche pragmatique

1. Choix d'une méthode d'ordonnancement convenant au problème.
2. Choix d'un type de communication convenant au problème.
3. Si la solution n'est pas optimale, itération vers une autre implémentation.

Nombres premiers.

1	2	3	4	5	6	7	8
T	T	T	F	T	F	T	F

1. Résultat: chaque processus en charge du calcul d'un élément (est_premier(i)).

2. Agenda: utilisation des résultats déjà disponibles.
   'Un nombre est premier s'il n'est divisible par aucun nombre premier strictement plus petit que sa racine carrée'.
   Si t est connu alors il est possible d'obtenir $[t+1,t^2]$.

3. Parallelism: Parcours de la structure à la recherche du nombre premier suivant.

Partie II : Linda

- Présentation générale
- Les opérateurs
- Les structures partagées

Introduction

- Invention du M.I.T (1982).
- **Philosophie** de coordination de processus distribués. Il en existe différentes implémentations : Linda-C, Prolog, Javaspaces…
- Un espace de mémoire composé de tuples (tuple space)

  Tuples = données composées (a, b), ('Linda', 'est', 'cool')
  - donnée: composant passif contenant une information.
  - processus: composant actif utilisant des tuples données. Lorsque le processus tuple n'est plus actif, il devient un 'Tuple données'.
- Quatre opérateurs.

Les opérateurs

OUT(t) : Ajoute le tuple t à l'espace.

```
OUT('Bruxelles', 'Uccle', 1180)
```

IN(s) : Retire de l'espace le tuple t correspondant aux critères du tuple s. L'opération est bloquante.

```
int i;
IN ('Bruxelles', 'Uccle', ? i)
'Le troisième composant doit être un entier'
```

Les opérateurs

RD(s): Lecture non destructive.

Identique que l'opérateur IN, sauf que le tuple t retrouvé reste dans l'espace.

EVAL(t)

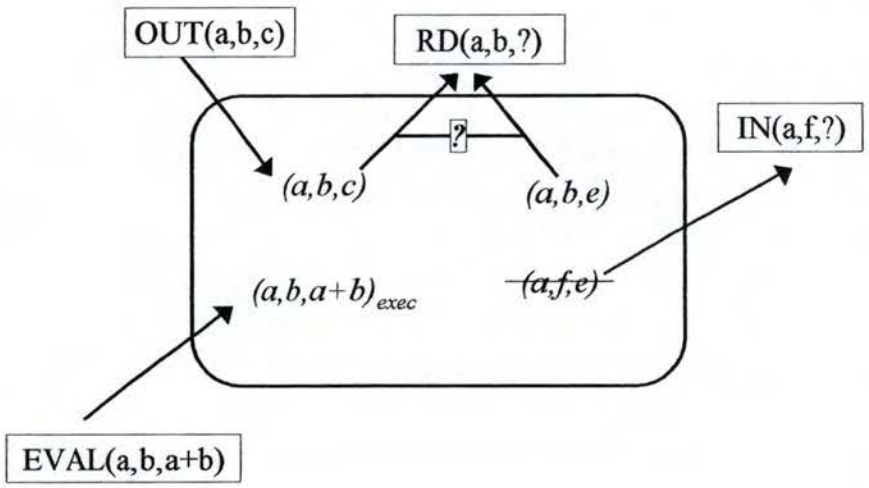
Création d'un tuple processus t.

```
Eval( 'sqr', 25, sqr(25))
```

Après évaluation, l'espace contient le tuple

```
('sqr', 25, 5)
```

Les opérateurs



Les structures.

- Sémaphore

  Création: `out('sémaphore')`

  Utilisation:

```
in('sémaphore')
exécution protégée
out(sémaphore)
```

Si besoin d'un accès concurrent avec maximum 3 instances :

```
out('sémaphore',1)
out('sémaphore',2)
out('sémaphore',3)
Avec in('sémaphore',? i)
```

Les structures

- Variables: `(nom, valeur)`

```
Écriture: out('PI', 3.14)
Lecture: rd('PI', ? val)
Update: in('PI', ? old)
        out('PI', 3.1415)
```

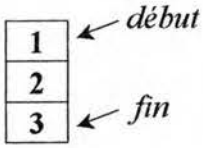
- Tableaux

```
in('A', 1, 1, 5.24)    in('A', 1, 2, 7.11)
in('A', 2, 1, 6.4)     in('A', 2, 1, 5.4)

for (x=1; x <=2; x++)
  for (y=1; y <=2; y++)
    rd("A", x, y, ? array[x,y])
```

Les structures

- File d'attente

```
out('liste', 1, val1)
out('liste', 2, val2)
out('liste', 3, val3)

out('liste','début',1)
out('liste','fin',3)
```



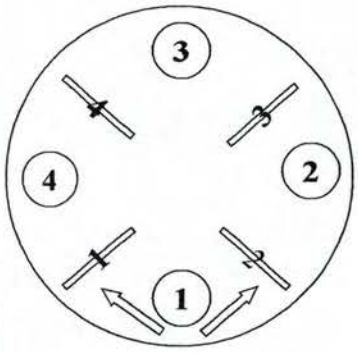
Insertion en fin de liste:

```
in ('liste', 'fin', ? index)
out('liste', 'fin', index+1)
out('liste', index, val4)
```

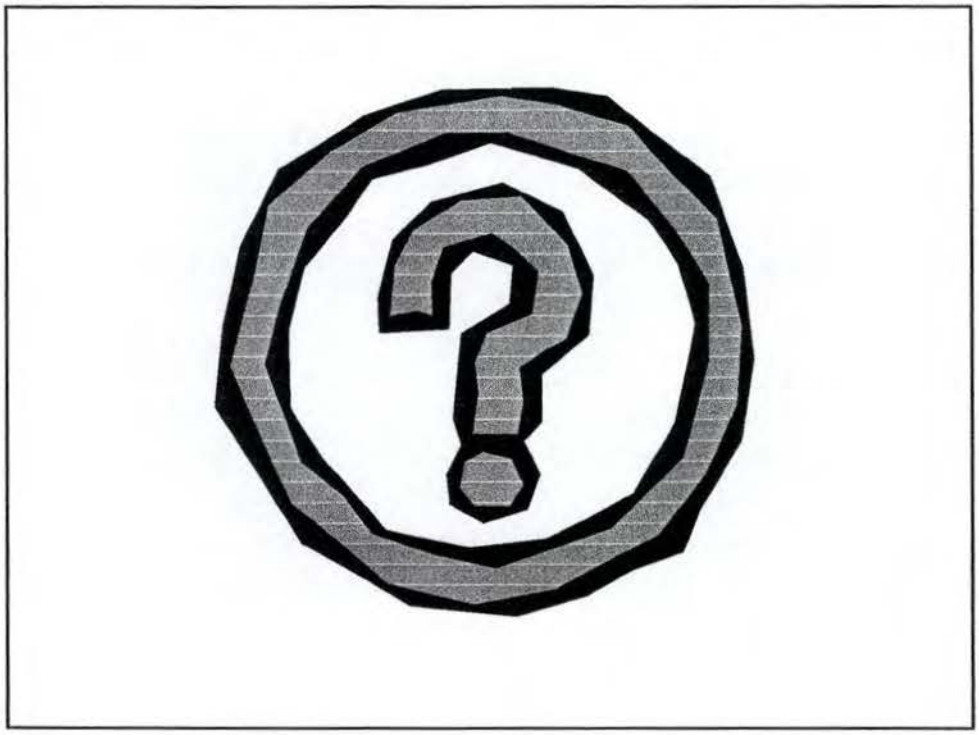
Suppression du 1er élément:

```
in ('liste', 'début', ? index)
out('liste', 'début', index+1)
in ('liste',  index, ? valeur)
```

Les philosophes.




```
init() {
  for (int i=0; i < Num; i++) {
    out('baguette', i);
    eval(philosophe(i));
    if (i < (Num-1)) out('ticket');
  }
}
philosophe( int i) {
  while(true) {
    think()
    in('ticket');
    in('baguette',i);
    in('baguette', (i+1)%Num);
    eat();
    out('baguette', i);
    out('baguette', (i+1)%Num);
    out('ticket'); }}
```



Partie III : Javaspaces

- Présentation générale et infrastructure
- Entry, Lease et Transactions
- Les opérateurs
- Notifications d'événements
- Installation
- Points de comparaison

Introduction

- Les processus coordonnent leurs exécutions en échangeant des objets Java (données passives) au travers d'un ***espace.***
- Javaspaces est ***librement*** inspiré par la philosophie Linda.

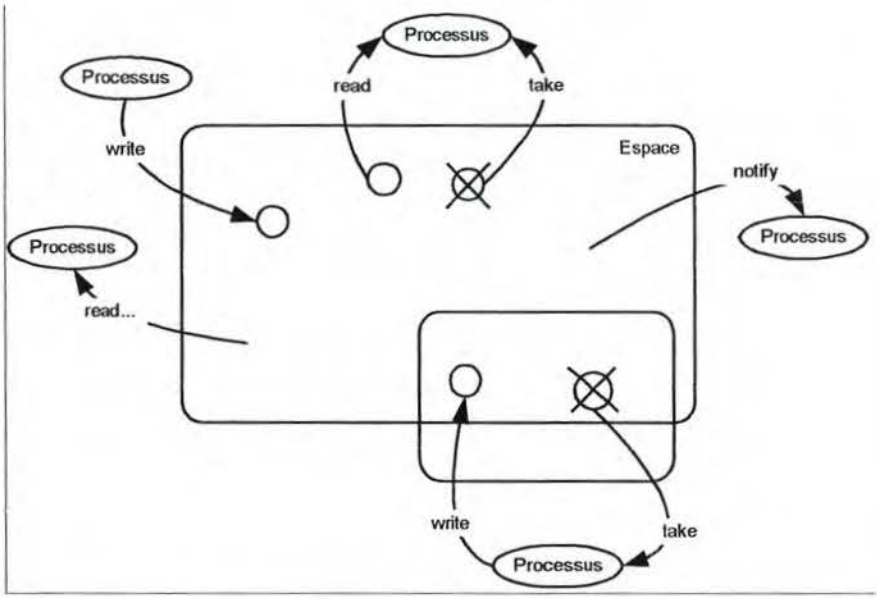
- **Espace**
  - Mémoire partagée
  - Partagé entre des processus
  - Persistant
  - Association par données (aucun d'identifiant)
  - Transactionnel (JINI)
  - Média de transport pour le code (HTTP)
  - Fortement décentralisée

Introduction

- Opérations sur l'espace :
  - **write** : Ecriture d'un objet
  - **read** : Lecture d'un objet
  - **take** : Lecture destructive d'un objet
  - **notify** : Notification

Chaque opération peut être liée à une transaction…

Introduction



Interface Entry

Objet Javaspaces => Entry

```
import net.jini.core.entry.Entry;

public class Msg implements Entry {
   public String  _From;
   public String  _To;
   public Integer _val;
   public Msg () {}
   public Msg (String _From,String _To, Integer -val) {…}
   public UneFonctionBienFoutue() { … }
}
```

- Propriétés publiques
- Constructeur sans arguments
- Sérialisation différente de Java !!!
- int => Integer

Interface Lease (JINI)

Permets de définir la durée de vie d'un objet Javaspaces.

```
public interface Lease {
   long FOREVER = Long.MAX_VALUE;
   long ANY = -1;

   long getExpiration();

   void cancel() …;
   void renew(long duration) …;

   LeaseMap createLeaseMap( long duration);
}
```

Interface JavaSpace

```
Lease write(Entry e, Transaction txn, long
  lease) throws RemoteException,
  TransactionException;

Entry read(Entry tmpl, Transaction txn, long
  timeout) throws TransactionException,
  UnusableException, RemoteException,
  InterruptedException;

Entry take(Entry tmpl, Transaction txn, long
  timeout) throws TransactionException,
  UnusableEntryException, RemoteException,
  InterruptedException;
```

Interface JavaSpace

```
Javaspaces espace = getSpace(…);
Transaction txn = …

Msg msg = new Msg('Toi','Moi', new Integer(5));
espace.write(msg, txn, Lease.FOREVER);

Msg tmpl = new Message();
Msg msg1, msg2;
tmpl._To = 'Moi';
tmpl._From = null;
tmpl._val=null;

msg1 = (Msg) espace.read(tmpl, txn, Long.MAX_VALUE);
msg2 = (Msg) espace.take(tmpl, txn, Long.MAX_VALUE);
```

Transactions (JINI)

- En cas de crash du client…
- Utilisable sur plusieurs espaces…
- WRITE: les tuples ne sont visibles que dans la transaction jusqu'au commit. En cas d'écriture sans transaction le tuple est disponible tout de suite aux clients de l'espace.
- READ: retrouve tous les tuples de sa transaction et tous les tuples n'étant pas repris dans une transaction.
- Abort automatique lorsque l'opération dépasse la durée de vie de la transaction.

Le serveur de tâches

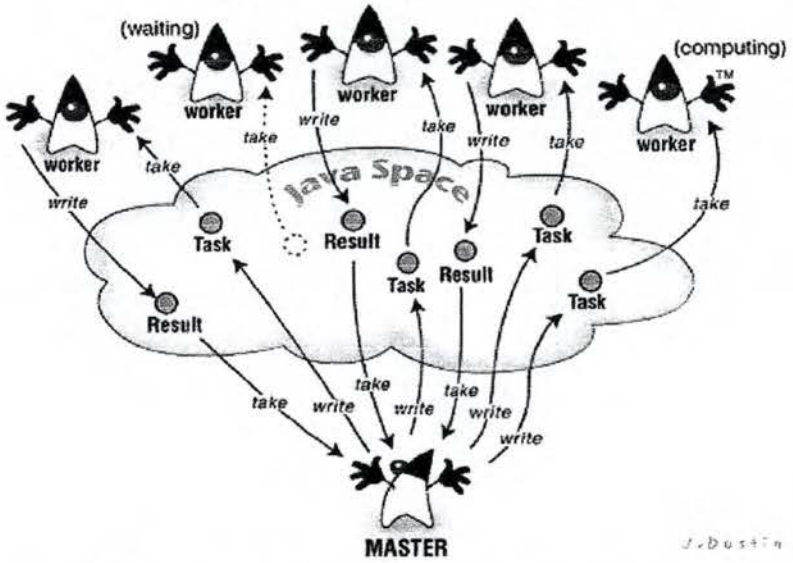
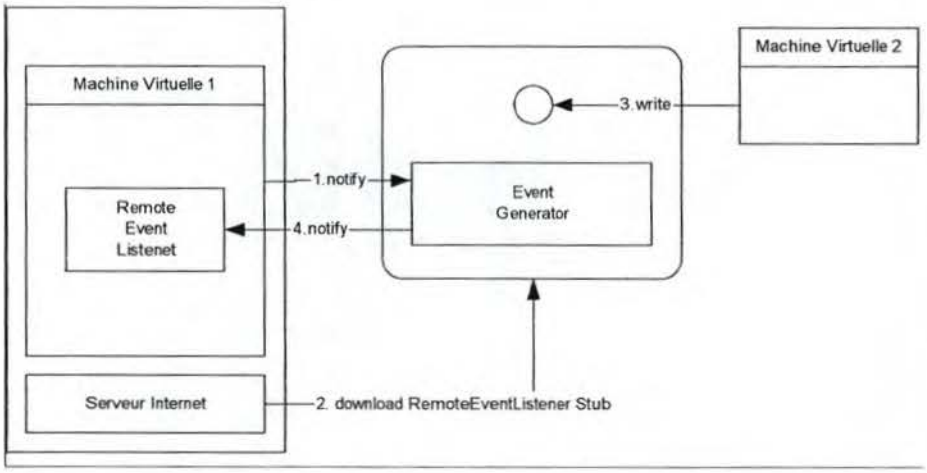


Illustration by James P. Dustin, Dustin Design

Le serveur de tâches: Démo

Notify

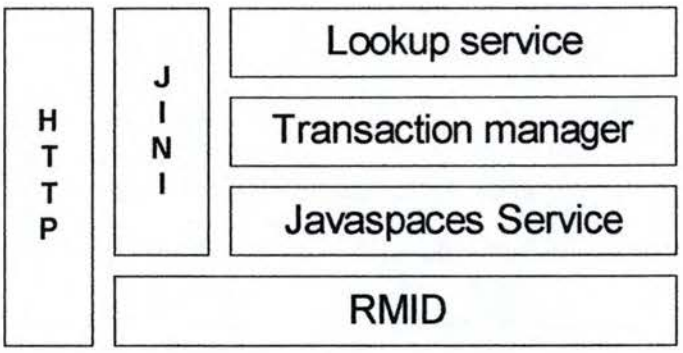


Notify

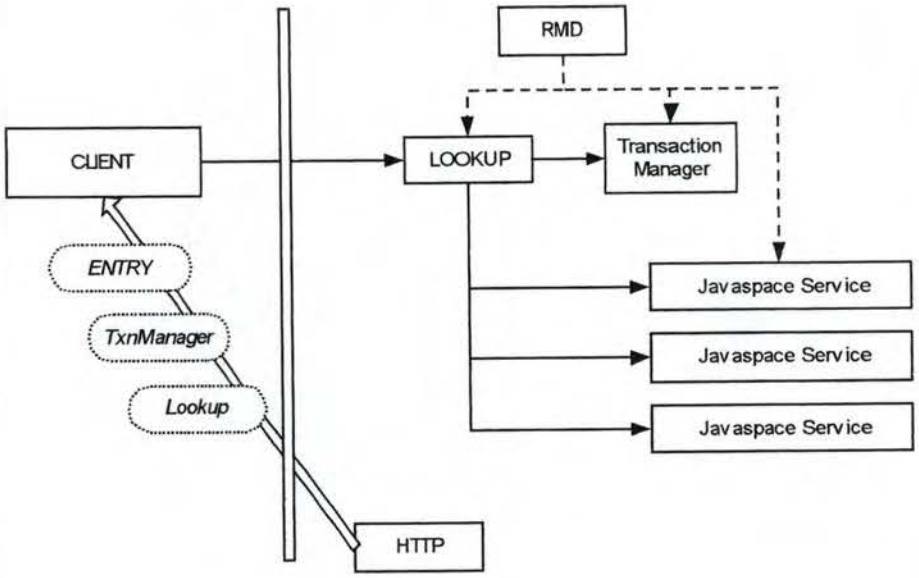
```
EventRegistration notify ( Entry tmpl,
                           Transaction txn,
                           RemoteEventListener listener,
                           long lease,
                           MarshalledObject handback)
  throws RemoteException, TransactionException;

public class EventRegistration implements Serializable {
  public long getId();
  public Object getSource();
  public long getSequenceNumber();
  public Lease getLease();
}

public interface RemoteEventListener {
  void notify(RemoteEvent theEvent) throws
       unknonwEventException, RemoteException;
}
```

Notify: Démonstration

Infrastructure



Infrastructure



Installation

- Java JDK 2, Standard Edition
- Jini starter kit
- Serveur Internet: Apache, IIS, ...

Installation

```
*-------- RMID

rmid.exe -log %log_rmid% -J-Djava.security.policy=%policy%

*------ Web Server

start java -jar %tools_jar% -port %port% -dir %jini_lib%

*------ LookUp Service

java -Djava.security.policy=%policy% -jar %reggie_jar%
%http%/reggie-dl.jar %policy% %log_lookup% %lookup_grp%

*------ Transaction Server

java -jar -Djava.security.policy=%policy% -
Dcom.sun.jini.mahalo.managerName=TransactionManager
%mahalo_jar% %http%/mahalo-dl.jar %policy% %log_transaction%
%lookup_grp%
```

Installation

```
*------ Javaspace Server

java -jar -Djava.security.policy=%policy%
     -Dnet.jini.discovery.interface=%IP%
     -Dcom.sun.jini.outrigger.spaceName=%space_general%
              %transient_jar% %http%/outrigger-dl.jar
     %policy%
     %log_space_general%
     %lookup_grp%
```

Les philosophes : démonstration

Gestion d'une transactions

```
TransactionManager TxnManager = null;
Transaction.Created txnC = null;
Transaction txn = null;
Lease lease = new Lease(…);

TxnManager = (TransactionManager)
   ServiceLocator.getService(TransactionManager.class);

txnC = TransactionFactory.create(TxnManager,lease);

txn = txnC.transaction;

txn.commit();
txn.abort();
```

Javaspaces vs Linda

Données fortement typées (type-safe). new T('a','b','c') <> new T('a','b','c').	Deux tuples ayant les même données sont identiques.
Des comportements (méthodes) peuvent être associées à tuple.	Pas de comportements associables à tuple.
Les tuples se retrouvent dans un ensemble de services JavaSpaces.	Un seul environnement pour tous les tuples.
Tuples associés avec une durée de vie (Lease).	Tuples doivent être supprimés manuellement.
N * write d'un même tuple = N tuples dans l'espace.	N * write d'un même tuple = 1 tuple dans l'espace.
Impossibilité à un client d'exécuter du code sur le serveur (EVAL) pouvant causer un problème de sécurité.	Eval()....

Javaspaces vs Base de données

Tuples....	Enregistrement...
Collection d'objets non structurés.	Collection d'objets structuré en table.
Recherche par critère.	Recherche par expressions complexes (SQL).
Services fortement distribués.	Client-Serveur.
Définition de la durée de vie par tuples.	Durées de vie infinie.
Indépendant de la plate-forme.	Dépendant de la plate-forme.
Transactions JINI utilisables par de multiples espaces ou d'autres services JINI !	Transactions propriétaires.
Opérations uniques pour le traitement des tuples.	Requiert la spécification par table pour Insert, Update, ...

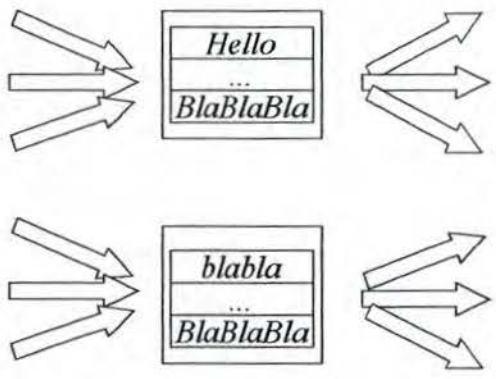
Avantages/Inconvénients de Javaspaces

- Performance: architecture très lourde
- Documentation
- Développement: Faible évolutivité du produit
- Conception: Oblige une mauvaise conception objet
- Evolution: Problème de la perte de propriétés

+ Facilité d'apprentissage
+ Nouvelles implémentations : Javabeans, GigaSpaces
+ Java
+ Intégration: Java , JSP, Beans, …

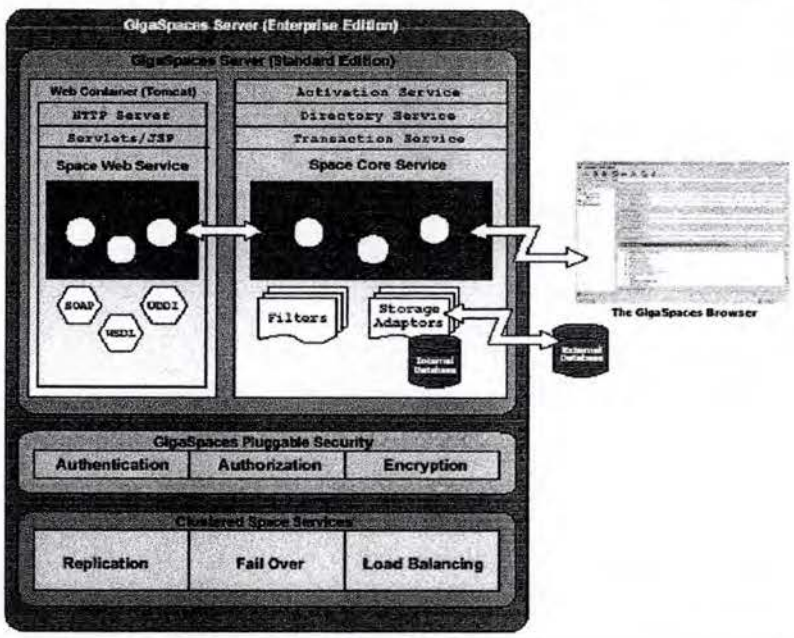
Un petit dernier pour la route: Intégration avec Java…

Gestion d'une file de messages pour un serveur chat.



Un petit dernier pour la route: Intégration avec Java…

GigaSpaces: l'alternative professionnelle ?



Références

- ***How to write parallel programs (a first course).***
  Nicholas Carriero – David Gelernter
  The MIT Press.
- ***Javaspaces Principles, Patterns, and Practice.***
  ISBN: 0-201-30955-6
  Freeman – Hupfer – Arnold
- ***JavaSpaces in practice.***
  ISBN: 0-321-11231-8
  Philip Bishop and Nigel Warren ().
- ***Make Room for JavaSpaces.***
  Eric Freeman

http://www.javasoft.com/javaspaces
http://www.jini.org

Les Philosophes

```
package IAMondeCommFwk.Formation.Philosophe;

import net.jini.core.entry.Entry;

public class _Baguette implements Entry {
  public Integer iID;

  /**
   * Instanciation à vide d'un baguette
   */
  public _Baguette() {}

  /**
   * Instanciation d'une baguette
   * @param iID identification du numéro de baguette
   */
  public _Baguette(int iID) { this.iID = new Integer(iID); }

}
```

```java
package IAMondeCommFwk.Formation.Philosophe;

import net.jini.core.entry.Entry;

public class _Ticket implements Entry {
  public String strNom = "Ticket";

  /**
   * Instanciation d'un Ticket d'accès au restaurant avec le nom par défaut.
   */
  public _Ticket() {}

  /**
   * Instanciation d'un Ticket d'accès au restaurant avec le nom par défaut.
   * @param strNom Identifie le Ticket
   */
  public _Ticket(String strNom) {
    this.strNom = strNom;
  }
}
```

```
package IAMondeCommFwk.Formation.Philosophe;

import net.jini.core.lease.Lease;
import net.jini.core.transaction.*;
import java.util.Random;
import net.jini.space.JavaSpace;
import IAMondeCommFwk.Espaces.CEspace;

public class Philosophe {
  /**
   * Constante identifiant le nombre de philosophes participant au repas
   */
  public static int cNum = 4;

  private CEspace oEspace;

  /**
   * Point de départ pour le repas.
   * @param args localisation de l'espace à utiliser
   */
  public static void main(String[] args) {
    if ( args.length != 1 ) {
      System.out.println("Utilisation: <URL>" +
                         "jini://lookup host/container name/JavaSpaces" );
      System.exit(1);
    }

    /**
     * Démarre le repas.
     */
    (new Philosophe(args[0])).initialize();
  }

  /**
   * Constructeur d'une application Philosophe.
   * @param strNomEspace
   */
  public Philosophe(String strNomEspace) {
    try {
      /* Initialisation de l'accès vers l'espace utilisé */
      oEspace = new CEspace(strNomEspace);

      /* Ouverture de la connection */
      oEspace.Connection();
    }
    catch (Exception e) { e.printStackTrace(); }
  }

  public void initialize() {
    int i;

    try {

      // Pour le Philosophe i
      for (i=0; i<cNum; i++) {
        // Ecriture dans l'espace de la baguette i
        _Baguette oBaguette = new _Baguette(i);
        oEspace.write(oBaguette, null, Lease.FOREVER) ;

        // Création du philosophe i.
        _Philosophe philosophe = new _Philosophe(oEspace.getNomEspace(), i);
        philosophe.start();

        // Ecriture du ième Ticket dans l'espace
        if (i < (cNum-1)) {
          oEspace.write(new _Ticket(),null,Lease.FOREVER);
        }
      }
    }
    catch (Exception e) { e.printStackTrace(); }
  }
}

class _Philosophe extends Thread {
  /** Identifie le philosophe */
```

```
public int iID = 0;

private Random rand = new Random();
private CEspace oEspace;

/**
 * Initialisation d'un philosophe
 * @param strNomEspace identifie l'espace
 * @param iID Identifie le philosophe
 */
public _Philosophe(String strNomEspace, int iID) {
  super();
  this.iID = iID;

  try {
    oEspace = new CEspace(strNomEspace);
    oEspace.Connection();
  }
  catch (Exception e) { e.printStackTrace(); }
}

/**
 * Démarrage d'un philosophe
 */
public void run() {
  System.out.println("Philosophe [" + String.valueOf(iID)+ "] started");
  Transaction t=null;

  try {

    while (true) {
      // ----- Le philosophe discute...
      this.think();

      // ----- Transaction d'une minute...
      t = oEspace.CreationTxn( (long) 60*1000*1);

      // ------ prendre un ticket d'accès
      _Ticket oTicket = (_Ticket) oEspace.take(new _Ticket(),t,Lease.FOREVER);
      System.out.println("Philosophe [" + String.valueOf(iID)+ "] entre dans la salle à manger" );

      // ------ prendre la baguette de gauche
      _Baguette oBaguette1 = (_Baguette) oEspace.take(new _Baguette(iID),t,Lease.FOREVER);

      // ------ prendre la baguette de droite d'accès
      _Baguette oBaguette2 = (_Baguette) oEspace.take(new _Baguette((iID+1)%Philosophe.cNum ),
                                                      t,
                                                      Lease.FOREVER);


      // ------ fin de transaction
      t.commit(); t=null;

      // ------ le philosophe mange...
      this.eat();

      // ------ Transaction d'une minute
      t = oEspace.CreationTxn(60*1000*1);

      // ----- Replacer dans l'espace des deux baguettes et du ticket d'accès
      oEspace.write(oBaguette1,t,Lease.FOREVER);
      oEspace.write(oBaguette2, t,Lease.FOREVER);
      oEspace.write(oTicket,t,Lease.FOREVER);

      // ------ fin de transaction
      t.commit(); t=null;

      System.out.println("Philosophe [" + String.valueOf(iID)+ "] quitte la salle à manger" );
    }
  }
  catch (Exception e) {
    e.printStackTrace();

    // Fin de transaction -> Rollback
    try { if (t != null) { t.abort(); } }
    catch (Exception e2) { e2.printStackTrace(); }
```

```
    }
  }

  /**
   * Le philosophe pense....
   */
  public void think() {
    try {
      long lThink = (int) (this.rand.nextDouble()*5000); // [0..5000]
      System.out.println("Philosophe [" + String.valueOf(iID)+ "] pense: [" +
                         String.valueOf(lThink)+"] msec");
      this.sleep(lThink);
    }
    catch (Exception e) { e.printStackTrace(); }
  }

  /**
   * Le philosophe mange...
   */
  public void eat() {
    try {
      long lEat = (int) (this.rand.nextDouble()*5000); // [0..2500]
      System.out.println("Philosophe [" + String.valueOf(iID)+ "] mange: [" +
                         String.valueOf(lEat)+"] msec");
      this.sleep(lEat);
    }
    catch (Exception e) { e.printStackTrace(); }
  }
}
```

Le serveur de tâches

```
package IAMondeCommFwk.Formation.TaskServer;

/**
 * Entry demandant le calcul d'une somme -> (a+b)
 */

import net.jini.core.entry.Entry;

public class _Addition implements Entry {

  public Integer a; // Premier paramètre de l'addition
  public Integer b; // Second paramètre de l'addition

  /**
   * Constructeur à vide d'une addition
   */
  public _Addition() {}

  /**
   * Constructeur avec paramètres pour l'addition a + b
   * @param a
   * @param b
   */
  public _Addition(Integer a, Integer b ) {
    this.a=a;
    this.b=b;
  }

  /**
   *
   * @return instance de _Resultat contenant le (a,b,a+b)
   */
  public _Resultat calcul() {

    int iResultat = a.intValue() + b.intValue();

    return new _Resultat( a, b, new Integer(iResultat));
  }

}
```

```
package IAMondeCommFwk.Formation.TaskServer;

import net.jini.core.entry.Entry;

/**
 *
 * Entry contenant le résultat d'une addition c = a+b
 */

public class _Resultat implements Entry {
  public Integer a = new Integer(0);
  public Integer b = new Integer(0);
  public Integer c = new Integer(0);

  /**
   * constructeur avec les paramètres par défaut
   */
  public _Resultat() {}

  /**
   * constructeur avec paramètres
   * @param a premier paramètre
   * @param b second paramètre
   * @param c = a+b
   */
  public _Resultat(Integer a, Integer b, Integer c) {
    this.a=a;
    this.b=b;
    this.c=c;
  }

  public String toString() {
    return a + " plus " + b + " = " + c;
  }
}
```

```
package IAMondeCommFwk.Formation.TaskServer;

import net.jini.core.entry.*;
import net.jini.core.lease.*;
import net.jini.core.transaction.*;
import net.jini.space.*;

import java.util.Random;
import IAMondeCommFwk.Espaces.*;

/**
 * Client réceptionnant le résultat des taches processées par le serveur.
 */

public class Client {
  public static int i;
  public static _Resultat oResult;

  public static void main(String[] args) {

    if ( args.length != 1 ) {
      System.out.println("Utilisation: <URL>" +
                         "jini://lookup host/container name/JavaSpaces" );
      System.exit(1);
    }

    try {
      // ----- Connection à l'espace.
      CEspace oEspace = new CEspace(args[0]);
      oEspace.Connection();

      // ----- Masque pour la recherche d'un résultat
      _Resultat oMasque = new _Resultat();

      for(;;) {
        // ----- Lecture d'un résultat
        oResult = (_Resultat) oEspace.take(oMasque,null,Long.MAX_VALUE);

        // ----- Affichage de la valeur du résultat à l'écran
        System.out.println(oResult.toString());
      }
    } catch (Exception e) {e.printStackTrace(); }
  }
}
```

```
package IAMondeCommFwk.Formation.TaskServer;

import net.jini.core.entry.*;
import net.jini.core.lease.*;
import net.jini.core.transaction.*;
import net.jini.space.*;

import java.util.Random;
import IAMondeCommFwk.Espaces.*;

/**
 * Serveur processant les taches disponibles dans l'espace.
 */

public class Serveur {
  public static JavaSpace _Espace;
  public static int i;

  public static void main(String[] args) {

    if ( args.length != 1 )  {
      System.out.println("Utilisation: <URL>"  +
                         "jini://lookup host/container name/JavaSpaces"  );
      System.exit(1);
    }

   try {
     // ----- Connection à l'espace
     CEspace oEspace = new CEspace(args[0]);
     oEspace.Connection();

     // ----- Définition du masque pour retrouver une addition
     _Addition oMasque = new _Addition();

     for (;;) {
       try {
         // ----- Prise d'une tâche dans l'espace
         _Addition oCalc = (_Addition) oEspace.take(oMasque, null, Long.MAX_VALUE);

         System.out.println("Calcul du résultat de l'opération "   + oCalc.a + " + " + oCalc.b );
         _Resultat result = (_Resultat) oCalc.calcul();

         // ----- écriture du résultat de a+b dans l'espace
         oEspace.write(result, null, 1000*60*10); // 10 min
       } catch (Exception e) {}
     }
   }
    catch (Exception e) { e.printStackTrace(); }
  }
}
```

```
package IAMondeCommFwk.Formation.TaskServer;

import net.jini.core.lease.*;
import net.jini.core.transaction.*;
import net.jini.space.*;

import java.util.Random;
import IAMondeCommFwk.Espaces.*;


/**
 *
 * Application permettant l'écriture de N taches dans l'espace.
 * Le nombre de taches est spécifiés en paramètre d'entrée.
 */
public class Taches {

  public static void main(String[] args) {
    int i;
    int iTaches = 0;

    if ( args.length != 2 ) {
      System.out.println("Utilisation: <URL> <Taches>"  +
                         " -> jini://lookup host/container name/JavaSpaces"    +
                         " -> spécifie le nombre de tâches à traiter"  );
      System.exit(1);
    }

    try {
      iTaches = Integer.valueOf(args[1]).intValue();
    }
    catch(NumberFormatException  e) {
      System.out.println("Utilisation: <URL> <Taches>"  +
                         "<Taches> doit être un nombre entier !!!"  );
    }

    try {
      // ----- Connexion à l'espace
      CEspace oEspace = new CEspace(args[0]);
      oEspace.Connection();

      // ----- Ecriture de n taches dans l'espace
      for (i=0; i< iTaches; i++) {
        oEspace.write(new _Addition(new Integer(i),new Integer(i)),
                      null,
                      Lease.FOREVER);
      }
    }
    catch (Exception e) { e.printStackTrace(); }
  }
}
```

Le canal de communications

```
package IAMondeCommFwk.Formation.Channel;

/**
 *  Entry définissant un canal de communication.
 */

import net.jini.core.entry.Entry;

public class _Channel implements Entry {

  public String strChannel;  // Nom du channel utilisé
  public Integer iPos;       // position du dernier message écrit.

  /**
   * Constructeur du channel avec les paramètres par défaut.
   */
  public _Channel() {}

  /**
   * Constructeur du channel avec paramètres.
   * @param strChannel nom du channel
   */
  public _Channel(String strChannel, Integer iPos) {
    this.strChannel = strChannel;
    this.iPos = iPos;
  }

  public _Channel(String strChannel) {
   this.strChannel = strChannel;
   this.iPos = new Integer(0);
 }


  /**
   * Demande de la dernière position utilisée par un message
   * @return la dernière position utilisée par un message
   */
  public Integer getPosition() {
    return iPos;
  }

  /**
   * Incrémente la position du dernier message ecrit
   */
  public void increment() {
    iPos = new Integer(iPos.intValue()+1);
  }

}
```

```
package IAMondeCommFwk.Formation.Channel;

/**
 * Entry définissant un Message contenu dans un canal de communication.
 */
import net.jini.core.entry.Entry;

public class _Message implements Entry {

  public String strChannel; // Nom du channel utilisé
  public Integer iPos;      // Position du message
  public String strContenu; // Texte du message

  /**
   * Constructeur avec les paramètres par défaut.
   */
  public _Message() {}

  /**
   * Constructeur paramétrisé
   * @param strChannel  nom du channel utilisé
   * @param iPos  position du message
   * @param strContenu texte du message
   */
  public _Message(String strChannel, Integer iPos, String strContenu) {
    this.strChannel = strChannel;
    this.iPos = iPos;
    this.strContenu = strContenu;
  }
}
```

```
package IAMondeCommFwk.Formation.Channel;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

import net.jini.core.transaction.*;
import net.jini.space.JavaSpace;
import IAMondeCommFwk.Espaces.*;
import net.jini.core.entry.Entry;
import net.jini.space.JavaSpace;

/**
 * Interface utilisateur permettant d'accéder aux fonctionnalités d'un canal
 * de communication
 */

public class ChannelGui extends Thread implements ActionListener  {

  private ChannelOperation oChnlOpe = null;

  JLabel lblChannel  = new JLabel("Channel:",JLabel.RIGHT );
  JLabel lblMessage  = new JLabel("Message:",JLabel.RIGHT);
  JLabel lblMessages = new JLabel("Messages:",JLabel.RIGHT);

  JTextField txtChannel  = new JTextField(25);
  JTextField txtMessage  = new JTextField(25);
  JTextArea txtMessages  = new JTextArea(15,40);

  JButton cmdLectureChannel = new JButton("Lecture");
  JButton cmdAddChannel = new JButton("Ajout");
  JButton cmdEnvoi = new JButton("Envoi");

  JFrame f = new JFrame();

  JPanel pnlMain = new JPanel();
  JPanel pnlChannel = new JPanel();
  JPanel pnlMessage = new JPanel();
  JPanel pnlMessages = new JPanel();

  final String cCreation = "Creation";
  final String cLecture  = "Lecture";
  final String cEcriture = "Ecriture";

  public static void main(String[] args) {

    if ( args.length != 1 ) {
      System.out.println("Utilisation: <URL>" );
      System.out.println("jini://lookup host/container name/JavaSpaces"  );
      System.exit(1);
    }
    ChannelGui app = new ChannelGui(args[0]);
  }

  public ChannelGui(String strNomEspace) {

    oChnlOpe = new ChannelOperation(strNomEspace);

    pnlMain.setLayout(new BorderLayout());

    cmdLectureChannel.addActionListener(this);
    cmdLectureChannel.setActionCommand(cLecture);

    cmdAddChannel.addActionListener(this);
    cmdAddChannel.setActionCommand(cCreation);

    cmdEnvoi.addActionListener(this);
    cmdEnvoi.setActionCommand(cEcriture);

    pnlChannel.setLayout(new FlowLayout(FlowLayout.LEFT));
    pnlChannel.add(lblChannel);
    pnlChannel.add(txtChannel);
    pnlChannel.add(cmdLectureChannel);
    pnlChannel.add(cmdAddChannel);
```

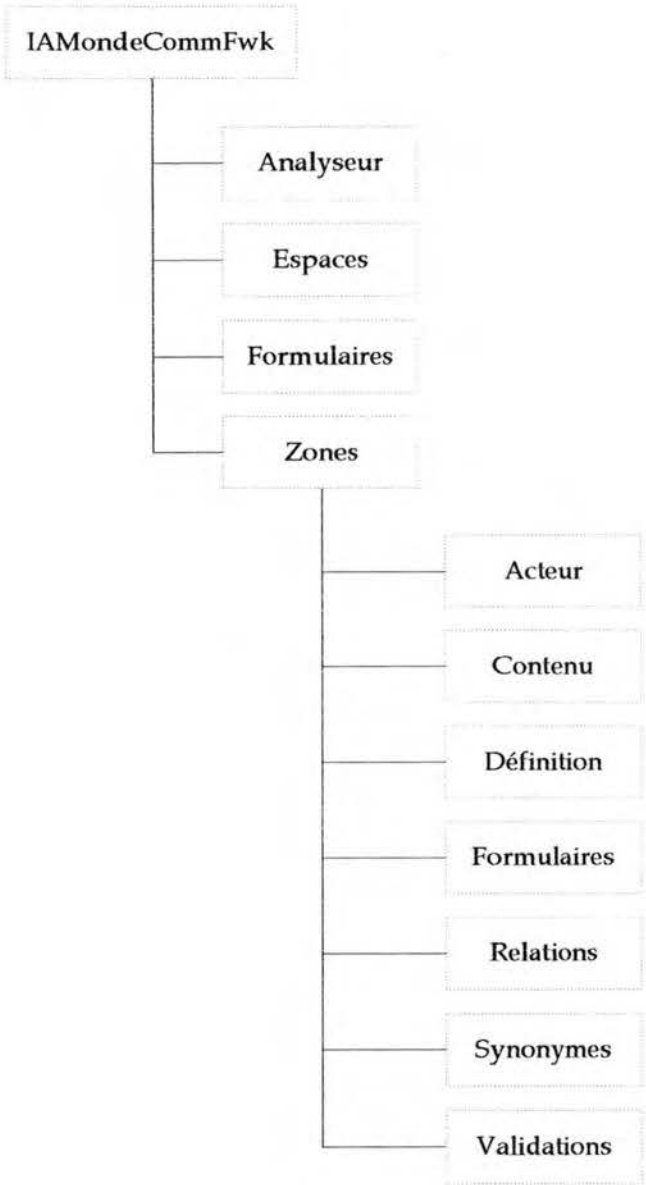
```
    pnlMessage.setLayout(new FlowLayout(FlowLayout.LEFT));
    pnlMessage.add(lblMessage);
    pnlMessage.add(txtMessage);
    pnlMessage.add(cmdEnvoi);

    pnlMessages.setLayout(new FlowLayout(FlowLayout.LEFT));
    pnlMessages.add(lblMessages);
    pnlMessages.add(txtMessages);

    pnlMain.add(pnlChannel,BorderLayout.NORTH);
    pnlMain.add(pnlMessage,BorderLayout.CENTER);
    pnlMain.add(pnlMessages,BorderLayout.SOUTH);

    f.addWindowListener(new BasicWindowMonitor());
    f.setContentPane(pnlMain);
    f.pack();
    f.setVisible(true);
  }

  public void actionPerformed(ActionEvent e) {
    String strCommand = e.getActionCommand();

    if (strCommand == cCreation) {
      this.oChnlOpe.CreationChannel(txtChannel.getText());
    }
    else if (strCommand == cLecture) {
      this.start();
    }
    else if (strCommand == cEcriture) {
      this.oChnlOpe.EcritureMessage(txtChannel.getText(),txtMessage.getText());
    }
  }

  public void run() {
    int iPosition = 1;

    try {
      while (true) {
        String strMessage = this.oChnlOpe.LectureMessage(txtChannel.getText(),iPosition);
        iPosition++;

        txtMessages.insert(strMessage+"\n",0);
      }
    }
    catch (Exception e) { e.printStackTrace(); }
  }
}

class BasicWindowMonitor implements WindowListener {
   public void windowActivated(WindowEvent e)   {}
   public void windowClosed(WindowEvent e)      {}
   public void windowClosing(WindowEvent e)     { System.exit(0); }
   public void windowDeactivated(WindowEvent e) {}
   public void windowDeiconified(WindowEvent e) {}
   public void windowIconified(WindowEvent e)   {}
   public void windowOpened(WindowEvent e)      {}
  }
```

```
package IAMondeCommFwk.Formation.Channel;

import IAMondeCommFwk.Espaces.*;
import net.jini.core.lease.Lease;

/**
 * Classe englobant les différentes opérations disponible sur un cannal
 * de communication.
 */

public class ChannelOperation {

  private CEspace oEspace=null;

  public ChannelOperation(String strNomEspace) {
    try {
      oEspace = new CEspace(strNomEspace);
      oEspace.Connection();
    }
    catch (Exception e) {e.printStackTrace(); }
  }

  public void CreationChannel(String strChannel) {
    try {
      System.out.println("CreationChannel: [" + strChannel + "]");
      _Channel oChannel = new _Channel(strChannel, new Integer(0));
      System.out.println("CreationChannel: [" + strChannel + "," + oChannel.getPosition() + "]");
      oEspace.write(oChannel, null, Lease.FOREVER);
    }
    catch (Exception e) { e.printStackTrace(); }
  }

  public String LectureMessage (String strChannel,int iPosition) {
    _Message oRet = null;

    System.out.println("LectureMessage: [" + strChannel + new Integer(iPosition).toString() + "]"
);
    _Message oMasque = new _Message(strChannel,
                                    new Integer(iPosition),
                                    null);

    try {
      oRet = (_Message) oEspace.read(oMasque,
                                     null,
                                     Long.MAX_VALUE);
    }
    catch (Exception e) {
      e.printStackTrace();

      oRet = new _Message( "",
                           new Integer(0),
                           "### Erreur générale lors de la lecture ###"   );
    }

    return oRet.strContenu;
  }


  public void EcritureMessage(String strChannel,String strMessage) {
    try {
      // ----- Réserve une position pour un message dans le channel
      System.out.println("EcritureMessage: [" + strChannel + "," + strMessage + "]");
      Integer iPosMsg = this.ReservePosition(strChannel);

      System.out.println("EcritureMessage position: [" + iPosMsg.toString() + "]");

      // ----- Ecriture du message dans le channel
      oEspace.write(new _Message(strChannel,iPosMsg,strMessage),
                    null,
                    Lease.FOREVER);
    }
    catch(Exception e) {e.printStackTrace();}
  }

  public Integer ReservePosition(String strChannel) {
```

```
    try {
      // ----- Lecture du Channel
      _Channel oChnl = new _Channel(strChannel,null);

      _Channel oRet = (_Channel) oEspace.take(oChnl,
                                              null,
                                              Long.MAX_VALUE);

      System.out.println("ReservePos: " + oRet.strChannel +"," + oRet.getPosition() );
      // ----- réserve une nouvelle position
      oRet.increment();

      // ----- écriture de Channel
      oEspace.write(oRet, null, Long.MAX_VALUE);

      // ----- retourne la nouvelle position disponible
      return oRet.getPosition();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
   return new Integer(0);
 }


}
```

Annexe 4 : Boîte à outils

Le présent code représente une implémentation des spécifications du chapitre 4. La boîte à outils est divisés en 'package Java' selon l'arborescence suivante :



Où :

- IAMondeCommFwk : l'ensemble des éléments globaux.
- Analyseur : les outils de gestion de l'analyseur sémantique.
- Espaces : les outils de gestion des espaces.
- Formulaires : les outils de gestion des formulaires.
- Zones : les outils de gestion des zones. Les sous packages étant les outils d'une zone spécifique.

Package : IAMondeCommFwk

```
package IAMondeCommFwk ;

/**
 * <p>Title: </p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author unascribed
 * @version 1.0
 */

import net.jini.core.entry.Entry;

public abstract class CConnaissance implements Entry {

  /* -- Clé unique de référence -- */
  private long  lIdentification =0;

  public CConnaissance () {}

  public long Identification () { return lIdentification ; }

  // représentation d'un acteur générique dans son domaine
  public String Description () {
    return "Connaissance de Base" ;
  }
}
```

Package : Analyseur

```
package IAMondeCommFwk.Analyseur;

import java.util.ArrayList;

import IAMondeCommFwk.Zone.Definition.EnumDefinition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Construction {

  private EnumDefinition enumDef;
  private Fonctions __Fonctions = null;

  public Construction() {
  }

  public Construction(EnumDefinition enumDef, Fonctions oFonctions) {
    this.setDefinition(enumDef = enumDef);
    this.setFonction(oFonctions);
  }

  public void setDefinition(EnumDefinition enumDef) { this.enumDef = enumDef;}
  public EnumDefinition getDefinition() { return this.enumDef; }

  public void setFonction(Fonctions oFonctions) { this.__Fonctions = oFonctions;}
  public Fonctions getFonctions() { return this.__Fonctions;}


}
```

```
package IAMondeCommFwk.Analyseur;

import java.util.*;
import java.util.ArrayList;
import IAMondeCommFwk.Zone.Definition.EnumDefinition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Constructions {

  private ArrayList __Constructions = new ArrayList();

  public Constructions() {
  }

  public void Ajouter(int iPosition, Construction constr) {
    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Ensemble vide -> insertion en position 1.
    if (__Constructions.isEmpty() ) { iPosition = 0; }

    // --- Ajout de l'élément à la position demandée.
    __Constructions.add(iPosition, constr);
  }

  public void AjouterFin(Construction ch) {
    // Ajout de l'élément à la fin.
    __Constructions.add(__Constructions.size(), ch);
  }

  public Iterator getConstructions() {
   return __Constructions.iterator();
 }

}
```

```
package IAMondeCommFwk.Analyseur;

import IAMondeCommFwk.Zone.Definition.EnumDefinition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Fonction {
   private EnumDefinition enumDef;

   private Fonctions __Fonctions = null;

   public Fonction() {
   }

   public Fonction(EnumDefinition enumDef) {
     this.setDefinition(enumDef = enumDef);
   }

   public void setDefinition(EnumDefinition enumDef) { this.enumDef = enumDef;}
   public EnumDefinition getDefinition() { return this.enumDef; }
}
```

```
package IAMondeCommFwk.Analyseur;

import java.util.*;
import java.util.ArrayList;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Fonctions {

  private ArrayList __Fonctions = new ArrayList();

   public Fonctions() {
   }

   public void Ajouter(int iPosition, Fonctions fct) {
     // --- Reposition sur Index 0 comme premier élément.
     iPosition--;

     // --- Ensemble vide -> insertion en position 1.
     if (__Fonctions.isEmpty() ) { iPosition = 0; }

     // --- Ajout de l'élément à la position demandée.
     __Fonctions.add(iPosition, fct);
   }

   public void AjouterFin(Fonctions fct) {
     // Ajout de l'élément à la fin.
     __Fonctions.add(__Fonctions.size(), fct);
   }

   public Iterator getConstructions() {
    return __Fonctions.iterator();
   }

 }
```

```
package IAMondeCommFwk.Analyseur;

import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.Definition.*;
import IAMondeCommFwk.Zone.Synonymes.*;
import IAMondeCommFwk.Zone.Definition.EnumDefinition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Mot {
  private String strMot="";
  private EnumDefinition enumDef;

  public Mot() {
  }

  public Mot(String strMot, EnumDefinition enumDef) {
    this.setDefinition(enumDef = enumDef);
    this.setMots(strMot);
  }

  public void setDefinition(EnumDefinition enumDef) { this.enumDef = enumDef;}
  public EnumDefinition getDefinition() { return this.enumDef; }

  public void setMots(String strMot) { this.strMot = strMot;}
  public String getMots() { return this.strMot;}


  public void Analyseur_Structuration_Creer (CEspace oDico) {
    CDefinition oDef = new CDefinition();
    CDefinition oDefRet = null;

    if (this.getDefinition().tolong() == EnumDefinition.Verbe) {
      oDef.InitGeneralFormeConjugee (this.getMots());
      try {
        oDefRet = (CDefinition) oDico.readIfExists(oDef, null, 1000 * 10);
      }
      catch (Exception e) { e.printStackTrace(); }

      if (null != oDefRet) {
        this.setDefinition(new EnumDefinition(oDefRet.lTypeDefinition.longValue() ));
      }
    }
  }

  /**
   * Projection de tous les verbes présent dans la phrase sur leur infinitif
   * @param oDico
   */

  public void Analyseur_Proj_Infinitif (CEspace oDico) {
    CDefinition oDef = new CDefinition();
    CDefinition oDefRet = null;

    if (this.getDefinition().tolong() == EnumDefinition.Verbe) {
      oDef.InitVerbeFormeConjugee (this.getMots());
      try {
        oDefRet = (CDefinition) oDico.readIfExists(oDef,null,1000*10);
      }
      catch (Exception e) {e.printStackTrace(); }

      if (null != oDefRet) { this.setMots(oDefRet.strDefinition);}
    }
  }

  /**
  * Projection de tous les mots présent dans la phrase sur leur synonymes
  * @param oDico
```

```
 */

 public void Analyseur_Proj_Synonyme (CEspace oDico) {
  Synonyme oSyn = new Synonyme();
  Synonyme oSynRet = null;

  oSyn.strMot_n = this.getMots();
  try {
      oSynRet = (Synonyme) oDico.readIfExists(oSyn, null, 100);
    }
    catch (Exception e) {e.printStackTrace(); }

    if (null != oSynRet) { this.setMots(oSynRet.strMot_0);}
  }
}
```

```
package IAMondeCommFwk.Analyseur;

import java.util.*;
import java.util.ArrayList;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Mots {
  private ArrayList __Mots = new ArrayList();

  public Mots() {
  }

  public void Ajouter(int iPosition, Mot oMot) {
    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Ensemble vide -> insertion en position 1.
    if (__Mots.isEmpty() ) { iPosition = 0; }

    // --- Ajout de l'élément à la position demandée.
    __Mots.add(iPosition, oMot);
  }

  public void AjouterFin(Mot oMot) {
    // Ajout de l'élément à la fin.
    __Mots.add(__Mots.size(), oMot);
  }

  public Iterator getIterator() {
    return __Mots.iterator();
  }

}
```

```
package IAMondeCommFwk.Analyseur;

import IAMondeCommFwk.Zone.Definition.EnumDefinition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Phrase {
   private EnumDefinition enumDef;
   private Mots __Mots = null;

   public Phrase() {
   }

   public Phrase(EnumDefinition enumDef, Mots oMots) {
     this.setDefinition(enumDef = enumDef);
     this.setMots(oMots);
   }

   public void setDefinition(EnumDefinition enumDef) { this.enumDef = enumDef;}
   public EnumDefinition getDefinition() { return this.enumDef; }

   public void setMots(Mots oMots) { this.__Mots = oMots;}
   public Mots getMots() { return this.__Mots;}

}
```

```
package IAMondeCommFwk.Analyseur;

import java.util.*;
import java.util.ArrayList;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.Definition.EnumDefinition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Phrases {

  private ArrayList __Phrases = new ArrayList();

  public Phrases() {
  }

  private void Ajouter(int iPosition, Phrase oPhrase) {
    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Ensemble vide -> insertion en position 1.
    if (__Phrases.isEmpty() ) { iPosition = 0; }

    // --- Ajout de l'élément à la position demandée.
    __Phrases.add(iPosition, oPhrase);
  }

  private void AjouterFin(Phrase oPhrase) {
    // Ajout de l'élément à la fin.
    __Phrases.add(__Phrases.size(), oPhrase);
  }

  /**
   * Voir spécification <B>analyseur_Phrase_Creer</B>
   * @param strPhrase ensemble d'Atomes.
   */
  public void Analyseur_Phrase_Creer (String strPhrase) {
    String Atomes[];
    Phrase oPhrase = new Phrase();
    Mots oMots = new Mots();

    Atomes = strPhrase.split("[ ,.;?!]");

    oPhrase.setDefinition(new EnumDefinition(EnumDefinition.Phrase ) );
    for (int i = 0; i < Atomes.length; i++) {
      if (!Atomes[i].equals("")) {
        oMots.AjouterFin(new Mot(Atomes[i].toLowerCase() ,
                                 new EnumDefinition(EnumDefinition.NonDefini )));
      }
    }
    oPhrase.setMots(oMots);

    this.AjouterFin(oPhrase);
  }

  /**
   * Voir la spécification <B>analyseur_Structuration_Créer</B>
   * @param oDico Espace contenant le dictionnaire
   */
  public void Analyseur_Structuration_Creer (CEspace oDico) {
    Iterator oPhrasesLst = this.getIterator();

    while (oPhrasesLst.hasNext()) {
      Phrase oPhrase = (Phrase) oPhrasesLst.next();

      Iterator oMotLst = oPhrase.getMots().getIterator();
      while (oMotLst.hasNext()) {
        Mot oMot = (Mot) oMotLst.next();
```

```
        oMot.Analyseur_Structuration_Creer (oDico);
      }
    }
  }

  /**
   * Affiche sous forme lisible la structure incluse dans Phrases...
   */
    public void Analyseur_Affiche_Structure () {
      String strStructure ="";
      Iterator oPhrasesLst = this.getIterator ();

      System.out.println(" _____ Structure de Phrases _____" );
      System.out.println(" ---------------------------------" );

      while (oPhrasesLst.hasNext()) {
        Phrase oPhrase = (Phrase) oPhrasesLst.next();

        System.out.println();
        System.out.println("__ Structure : "  + oPhrase.getDefinition().toString());
        Iterator oMotLst = oPhrase.getMots().getIterator();
        while (oMotLst.hasNext()) {
          Mot oMot = (Mot) oMotLst.next();
          System.out.println("    __ " + oMot.getMots() +
                             "(" + oMot.getDefinition().toString() + ")");
        }
      }
      System.out.println(" _____" );
      System.out.println(" ------ Fin de structure -----" );
      System.out.println();

    }


 /**
   * Voir la spécification <B>analyseur_Proj_Infinitif</B>
   * @param oDico Espace contenant le dictionnaire
   */
  public void Analyseur_Proj_Infinitif (CEspace oDico) {
    Iterator oPhrasesLst = this.getIterator ();

    while (oPhrasesLst.hasNext()) {
      Phrase oPhrase = (Phrase) oPhrasesLst.next();

      Iterator oMotLst = oPhrase.getMots().getIterator();
      while (oMotLst.hasNext()) {
        Mot oMot = (Mot) oMotLst.next();
        oMot.Analyseur_Proj_Infinitif (oDico);
      }
    }
  }

  /**
   * Voir la spécification <B>analyseur_Proj_Synonymes</B>
   * @param oDico Espace contenant le dictionnaire
   */
  public void Analyseur_Proj_Synonyme (CEspace oDico) {
    Iterator oPhrasesLst = this.getIterator ();

    while (oPhrasesLst.hasNext()) {
      Phrase oPhrase = (Phrase) oPhrasesLst.next();

      Iterator oMotLst = oPhrase.getMots().getIterator();
      while (oMotLst.hasNext()) {
        Mot oMot = (Mot) oMotLst.next();
        oMot.Analyseur_Proj_Synonyme (oDico);
      }
    }
  }

 /**
   * Voir la spécification <B>analyseur_Proj_Phrase</B>
   * Reconstitue une phrase en se basant sur la liste de mots
   */
  public String Analyseur_Proj_Phrase () {
```

```
    String strMots="";
    Iterator oPhrasesLst = this.getIterator();

    while (oPhrasesLst.hasNext()) {
      Phrase oPhrase = (Phrase) oPhrasesLst.next();

      Iterator oMotLst = oPhrase.getMots().getIterator();
      while (oMotLst.hasNext()) {
        Mot oMot = (Mot) oMotLst.next();
        if ( strMots.equals("")) {
          strMots = strMots + oMot.getMots().substring(0,1).toUpperCase();
          strMots = strMots + oMot.getMots().substring(1);
        }
        else {
          strMots = strMots + " ";
          strMots = strMots + oMot.getMots();
        }
      }
    }
    return strMots + ".";
  }


  public Iterator getIterator() {
    return __Phrases.iterator();
  }

}
```

Package : Espaces

```
package IAMondeCommFwk .Espaces;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class CEspaceManipulation  {

  /* --- Zones Acteur ------------------------------------------------------*/
  public final static String strZoneActeurCourriel    = "Acteur Courriel" ;
  public final static String strZoneActeurCourrier    = "Acteur Courrier" ;
  public final static String strZoneActeurHumain      = "Acteur Humain" ;
  public final static String strZoneActeurGenerique   = "Acteur Générique" ;
  public final static String strZoneTelephone         = "Acteur Téléphone" ;

  /* --- Zones Contenu -----------------------------------------------------*/
  public final static String strZoneContenuTextLibre  = "ZoneContenuTextLibre" ;

  /* --- Zones Formulaire --------------------------------------------------*/
  public final static String strZoneFormulaire  = "ZoneFormulaire" ;

  /* --- Zones Validation --------------------------------------------------*/
  public final static String strZoneValidationEstDate      = "ZoneValidationEstDate" ;
  public final static String strZoneValidationEstUnNombre  = "ZoneValidationEstUnNombre" ;
  public final static String strZoneValidationEstPasVide   = "ZoneValidationEstPasVide" ;
  public final static String strZoneValidationTailleMax    = "ZoneValidationTailleMax" ;

  /* --- Zones Fonctions ---------------------------------------------------*/
  public final static String strZoneConstructions  = "ZoneConstructions" ;
  public final static String strZoneSynonyme  = "ZoneSynonyme" ;
}
```

```
package IAMondeCommFwk.Espaces;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 *
 * Implémentation d'un ensemble de type Espace
 */

import net.jini.core.entry.*;
import net.jini.core.event.*;
import net.jini.core.lease.*;
import net.jini.core.transaction.*;
import net.jini.core.transaction.server.*;
import net.jini.space.*;
import java.rmi.*;
import com.j_spaces.core.*;

import IAMondeCommFwk.CConnaissance;
import IAMondeCommFwk.Zone.Validations.*;

import com.j_spaces.core.client.LocalTransactionManager;
import com.j_spaces.core.client.SpaceFinder;

public class CEspace {

  /* --- Nom de l'espace initialisé à une valeur par défaut --- */
  private String strNomEspace = "JavaSpaces";

  private IJSpace __Espace = null;

  public CEspace() {
    super();
    this.strNomEspace = this.strNomEspace;
  }

  public CEspace(String strNomEspace) {
    super();
    this.strNomEspace = strNomEspace;
  }

  /**
   * @return : le nom de l'Espace
   */
  public String getNomEspace() {
    return this.strNomEspace;
  }

  /**
   *
   * @param strNomEspace : le nom de l'Espace
   */
  public void setNomEspace(String strNomEspace) {
    this.strNomEspace = strNomEspace;
  }

  public IJSpace getEspace() { return this.__Espace; }

  /**
   *
   * @return
   *   true: l'accès vers l'Espace est initialisé
   */
  public boolean EspaceAccessible() {
    return this.__Espace != null;
  }

  /**
   * Ouverture de connection vers l'Espace
   * @throws Exception
   */
```

```
public void Connection () throws Exception {
  try {
    __Espace = (IJSpace) SpaceFinder.find( this.strNomEspace );
  }
  catch (Exception e) {
    throw new Exception("Exception :CEspace.Connection("  + this.strNomEspace + ")"
                        + e.toString() );
  }
}

public final Transaction CreationTxn(long lDuree) throws Exception {

  LocalTransactionManager  trManager;
  Transaction.Created      tCreated;

  try {
    // ------ creation d'un manager de transaction
    trManager = new LocalTransactionManager ((com.j_spaces.core.IJSpace) this.__Espace );

    // ------ creation de la transaction
    tCreated = TransactionFactory.create(trManager, lDuree);
    return tCreated.transaction;
  }
  catch (Exception e) {
    throw new Exception("Exception :CEspace.CreationTxn(" +String.valueOf(lDuree).toString()+")"
                        + e.toString() );
  }
}

/**
 * Implémentation de la primitive JavaSpaces : 'write'
 * @param entry
 * @param txn
 * @param lease
 */
public void write(Entry entry, Transaction txn, long lease) throws Exception{
  try { this.__Espace.write(entry, txn, lease); }
  catch (Exception e) {
    throw new Exception("CEspace.write() "  + e.toString() );
  }
}

/**
 * Implémentation de la primitive JavaSpaces : 'read'
 * @param tmpl
 * @param txn
 * @param lease
 * @return
 * @throws Exception
 */
public Entry read(Entry tmpl, Transaction txn, long lease) throws Exception {
  try {
    return  this.__Espace.read(tmpl, txn, lease);
  }
  catch (Exception e) {
    throw new Exception("CEspace.read() "  + e.toString() );
  }
}

public Entry readIfExists(Entry tmpl, Transaction txn, long lease) throws Exception {
   try {
     return  this.__Espace.readIfExists(tmpl, txn, lease);
   }
   catch (Exception e) {
     throw new Exception("CEspace.readIfExists() "  + e.toString() );
   }
 }

/**
 * Implémentation de la primitive JavaSpaces : 'take'
 * @param tmpl
 * @param txn
 * @param lease
 * @return
 * @throws Exception
```

```
   * @throws Exception
   */
  public Entry take(Entry tmpl, Transaction txn, long lease) throws Exception {
    try { return this.__Espace.take(tmpl, txn, lease); }
    catch (Exception e) {
      throw new Exception("CEspace.take() " + e.toString() );
    }
  }


  public Entry takeIfExist(Entry tmpl, Transaction txn, long lease) throws Exception {
   try {
     return this.__Espace.takeIfExists(tmpl, txn, lease);
   }
   catch (Exception e) {
     throw new Exception("CEspace.takeIfExists() " + e.toString() );
   }
  }



  /**
   * Implémentation de la primitive JavaSpaces : notify
   * @param entry
   * @param txn
   * @param listener
   * @param lease
   * @param obj
   * @throws Exception
   */
  public EventRegistration notify(Entry entry,
                    Transaction txn,
                    RemoteEventListener listener,
                    long lease,
                    MarshalledObject obj) throws Exception{
    try { return this.__Espace.notify(entry, txn, listener, lease, obj); }
    catch (Exception e) {
      throw new Exception("CEspace.notify() " + e.toString() );
    }

  }

}
```

Package : Formulaires

```
package IAMondeCommFwk.Formulaires;

import java.util.ArrayList;
import java.util.*;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 *
 * Implémentation d'un élément du type d'ensemble Chapitres.
 */

public class Chapitre {

  private String strIntitule = "";
  private Questions __Questions = null;

  public Chapitre() { }

  public Chapitre(String strIntitule) {
    this.strIntitule = strIntitule;
  }

  public Chapitre(String strIntitule, Questions oQuestions) {
    this.strIntitule = strIntitule;
    this.__Questions = oQuestions;
  }

  public void setIntitule(String strIntitule) { this.strIntitule = strIntitule;}
  public String getIntitule() { return this.strIntitule; }

  public void setQuestions(Questions oQuestions) { this.__Questions = oQuestions;}
  public Questions getQuestions() { return this.__Questions;}

  /**
   * Un Chapitre est valide si toutes ses questions sont valides.
   * @return true => le chapitre est valide, false => le chapitre n'est pas valide.
   */
   public boolean estValide() {
     boolean fValide = true;
     Iterator quest = __Questions.getQuestions();

     while (quest.hasNext()) {
       Question q = (Question) quest.next();
       if (!q.estValide()) { fValide = false; }
     }
     return fValide;
   }
}
```

```
package IAMondeCommFwk.Formulaires;

import java.util.ArrayList;
import java.util.*;


/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Chapitres {

  private ArrayList __Chapitres = new ArrayList();

  public Chapitres() { }

  public void Ajouter(int iPosition, Chapitre ch) {
    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Ensemble vide -> insertion en position 1.
    if (__Chapitres.isEmpty() ) { iPosition = 0; }

    // --- Ajout de l'élément à la position demandée.
    __Chapitres.add(iPosition, ch);
  }

  public void AjouterFin(Chapitre ch) {
    // Ajout de l'élément à la fin.
    __Chapitres.add(__Chapitres.size(), ch);
  }

  public void Enlever(int iPosition) {
    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Aucun élément à enlever.
    if (__Chapitres.isEmpty() ) { return; }

    // --- la position demandée dépasse le dernier élément.
    if (iPosition > __Chapitres.size() ) { return; }

    // --- suppression de l'élément demandé.
    __Chapitres.remove(iPosition);
  }

  public Question Editer(int iPosition) {

    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Envoi de l'élément demandé.
    return (Question) __Chapitres.get(iPosition);
  }

  public int Position(Chapitre ch) {
    // --- Recherche de l'élément demandé
    //   - Reposition sur Index 1 comme premier élément.
    int iPosition = __Chapitres.indexOf(ch)+1;

    // --- Si l'élément est retrouvé, envoi de sa position.
    return iPosition > 0 ? iPosition : 0;
  }

  public Chapitre Premier(Chapitre ch) {
    if (__Chapitres.isEmpty()) {
      return new Chapitre("");
    }
    else {
      return (Chapitre) __Chapitres.get(0);
```

```
    }
  }

  public Iterator getChapitres() {
    return __Chapitres.iterator();
  }

  public void decodeFormulaire(String strFormulaire) throws Exception {

    Chapitre start_chap, stop_chap;

    Iterator chap = __Chapitres.iterator();

    boolean fExistNext = true;

    String strDebutChapitre, strFinChapitre,strToDecode;
    int iStartPos, iStopPos;

    // ------ Découpage du texte par intitulé de chapitres
    stop_chap = (Chapitre) chap.next();
    while (fExistNext) {
      start_chap = stop_chap;
      strDebutChapitre = start_chap.getIntitule();

      // si le texte ne commence par l'intitué de Chapitre -> erreur.
      if (!strFormulaire.startsWith(strDebutChapitre)) {
        throw new Exception("Il manque un Chapitre !!!");
      }

      iStartPos = strDebutChapitre.length();

      // localise le chapitre suivant.
      if (chap.hasNext()) {
        stop_chap = (Chapitre) chap.next();
        strFinChapitre = stop_chap.getIntitule();

        if (-1 == (iStopPos = strFormulaire.indexOf(strFinChapitre))) {
          throw new Exception("Il manque un Chapitre !!!");
        }
      }
      else {
        fExistNext = false;
        iStopPos=strFormulaire.length();
      }

      // Texte à décoder pour le chapitre
      strToDecode = strFormulaire.substring(iStartPos,iStopPos);

      // Décodage des réponses.
      try {
        start_chap.getQuestions().decodeFormulaire(strToDecode);
      }
      catch (Exception e) { throw(e); }

      // Suppression du Chapitre traité du Formulaire.
      strFormulaire = strFormulaire.substring(iStopPos);

    }
  }

}
```

```
package IAMondeCommFwk.Formulaires;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>
 *   Description: Définition d'un élément du type d'ensemble Questions<BR>
 *                défini comme ( Intitulé, ensValidations )</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 *
 *
 *
 */

import java.util.*;
import IAMondeCommFwk.Zone.Validations.IValidations;

public class Question {

  private String strIntitule = "";
  private String strReponse = "";

  /**
   * Définition d'un ensemble non ordonné pour contenir ensValidations
   */
  private Validations __Validation = null;

  /**
   * Création d'un élément Question = ( "", {} )
   */
  public Question() { }

  /**
   * Création d'un élément Questions = ( Intitulé, {} )
   * @param strIntitule valeur de l'Intitulé de la question.
   */
  public Question(String strIntitule) {
    this.strIntitule = strIntitule;
  }

  /**
   *
   * Création d'un élément Questions = ( Intitulé, oValidation ) <BR>
   * @param strIntitule valeur de l'Intitulé de la question. <BR>
   * @param oValidation valeur pour le composant ensValidation
   */
  public Question(String strIntitule,Validations oValidation) {
    this.strIntitule = strIntitule;
    this.__Validation = oValidation;
  }

  /**
   * @return la valeur du composant <B><I>Intitulé</I><B> de l'élément Question
   */
  public String getIntitule() { return this.strIntitule; }

  /**
   * Définition d'un élément Reponse du type d'ensemble Réponses définit
   * comme (Chapitre, Question, Réponse) <BR>
   * avec : <BR>
   *   - Chapitre = s'applique au même chapitre que l'élément Question <BR>
   *   - Question = l'élément Question <BR>
   *   - Réponse = strReponse <BR>
   *
   * @param strReponse la valeur du composant <B><I>Réponse</I></B>
   */
  public void setReponse(String strReponse) { this.strReponse = strReponse; }

  /**
   * Soit un élément Reponse du type d'ensemble Réponses définit
   * comme (Chapitre, Question, Réponse) <BR>
   * avec : <BR>
   *   - Chapitre = s'applique au même chapitre chapitre que l'élément Question
```

```
   *  - Question = l'élément Question
   *
   * @return valeur du Composant Réponse
   */
  public String getReponse() { return this.strReponse; }

  /**
   * @param oValidation Nouvelle valeur pour le composant ensValidations de l'élément Question.
   */
  public void setValidation(Validations oValidation) {
    this.__Validation   = oValidation;
  }

  /**
    * @return la valeur du composant <B><I>ensValidations</I><B> de l'élément Question
    */
  public Validations getValidation() {
    return this.__Validation ;
  }

  /**
   * Méthode requise pour pouvoir comparer deux éléments Question
   *
   * @param oQuestion l'élément à comparer
   * @return true les deux éléments sont identiques
   */
  public boolean equals(Question oQuestion) {
    return this.strIntitule.equals(oQuestion.strIntitule);
  }

  /**
   * Une question est valide si toutes ses réponses répondent aux exigences
   * définies dans ensValidations.
   * @return true => la question est valide, false => la question n'est pas valide.
   */
  public boolean estValide() {
    boolean fValidation = true;
    boolean fValide = true;

    try {
      if (null != this.getValidation()) {
        Iterator valid = this.getValidation().ListeValidation();

        while (valid.hasNext() ) {
          fValidation =((IValidations) valid.next()).Validation(this.getReponse());

          if (!fValidation) {
            System.out.println("La question: " + this.getIntitule() +
                               "ne réponds pas à ses Validations" );
            fValide = false;
            break;
          }
        }
      }
    }
    catch (Exception e) {
      System.out.println("Question.estValide() "  + e.toString() );
      fValide = false;
    }

    return fValide;

  }

}
```

```
package IAMondeCommFwk.Formulaires;

import java.util.ArrayList;

import IAMondeCommFwk.*;
import java.util.*;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Questions {

  private ArrayList __Questions = new ArrayList();

  public Questions() { super(); }

  public void Ajouter(int iPosition, Question q) {

    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Ensemble vide -> insertion en position 1.
    if (__Questions.isEmpty() ) {
      iPosition = 0;
    }

    // --- demande d'insertion après le dernier élément.
    if (iPosition > __Questions.size() ) {
      iPosition = __Questions.size();
    }

    // --- Ajout de l'élément à la position demandée.
    __Questions.add(iPosition,q);
  }

  public void AjouterFin(Question q) {
    // Ajout de l'élément à la fin.
    __Questions.add(__Questions.size(), q);
  }

  public void Enlever(int iPosition) {
    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Aucun élément à enlever.
    if (__Questions.isEmpty() ) { return; }

    // --- la position demandée dépasse le dernier élément.
    if (iPosition > __Questions.size() ) { return; }

    // --- suppression de l'élément demandé.
    __Questions.remove(iPosition);
  }

  public Question Editer(int iPosition) {

    // --- Reposition sur Index 0 comme premier élément.
    iPosition--;

    // --- Envoi de l'élément demandé.
    return (Question) __Questions.get(iPosition);
  }

  public int Position(Question q) {
    // --- Recherche de l'élément demandé
    //   - Reposition sur Index 1 comme premier élément.
    int iPosition = __Questions.indexOf(q)+1;

    // --- Si l'élément est retrouvé, envoi de sa position.
```

```
    return iPosition > 0 ? iPosition : 0;
  }

  public Question Premier( Question q) {
    if (__Questions.isEmpty()) {
      return new Question("");
    }
    else {
      return (Question) __Questions.get(0);
    }
  }

  public Iterator getQuestions() {
    return __Questions.iterator();
  }

  public void decodeFormulaire(String strChapitre) throws Exception {

    Question start_quest, stop_quest;

    Iterator quest = __Questions.iterator();

    boolean fExistNext = true;

    String strDebutQuestion, strFinQuestion,strToDecode;
    int iStartPos, iStopPos;

    // ------ Découpage du texte par intitulé de chapitres
    stop_quest = (Question) quest.next();

    while (fExistNext) {
      start_quest = stop_quest;
      strDebutQuestion = start_quest.getIntitule();

      // si le texte ne commence par l'intitué de Chapitre -> erreur.
      if (!strChapitre.startsWith(strDebutQuestion)) {
        throw new Exception("Il manque une Question !!!");
      }

      iStartPos = strDebutQuestion.length();

      // localise le chapitre suivant.
      if (quest.hasNext()) {
        stop_quest = (Question) quest.next();
        strFinQuestion = stop_quest.getIntitule();

        if (-1 == (iStopPos = strChapitre.indexOf(strFinQuestion))) {
          throw new Exception("Il manque une Question !!!");
        }
      }
      else {
        fExistNext = false;
        iStopPos=strChapitre.length();
      }

      // Texte à décoder pour le chapitre
      strToDecode = strChapitre.substring(iStartPos,iStopPos);

      // Ajout de la réponse
      try {
       System.out.println(strDebutQuestion + " : " + strToDecode);
       start_quest.setReponse(strToDecode);
      }
      catch (Exception e) { throw(e); }

      // Suppression de la Question traitée du Chapitre.
      strChapitre = strChapitre.substring(iStopPos);

    }

  }
}
```

```
package IAMondeCommFwk.Formulaires;

import java.util.*;
import IAMondeCommFwk.Zone.Validations.IValidations;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */


public class Validations {

  /* contient la liste des validations */
  private HashSet __Validations = null;

  public Validations() { __Validations = new HashSet();}

  /**
   * Ajoute un élément dans la liste des validation?
   *
   * @param validation
   *   Validation implémentant l'interface IValidations
   *
   * @throws Exception
   *   Erreur lors de l'exécution de la méthode.
   */
  public void Ajout(IValidations validation) throws Exception {
    try {
      this.__Validations.add(validation);
    }
    catch (Exception e) {
      e.printStackTrace();
      throw (new Exception("Erreur lors de Validations.Ajout()" ));
    }
  }

  public Iterator ListeValidation() throws Exception {

    Iterator liste=null;

    try {
      liste = this.__Validations.iterator();
    }
    catch (Exception e) {
      e.printStackTrace();
      throw (new Exception("Erreur lors de Validations.ListeValidation()" ));
    }

    return liste;
  }
}
```

Package : Zone

```
package IAMondeCommFwk .Zone;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

/**
 * IZone est l'Interface de définition d'un ensemble Zone
 *
 */
public interface IZone {
  String zone_Identification ();
  String toString ();
}
```

Package : Acteur

```
package IAMondeCommFwk.Zone.Acteur;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CActeurCourriel extends CConnaissance implements IZone {

  // --- Interface publique disponible dans l'espace
  public String strNom;
  public String strServeur;
  public String strDomaine;

  public CActeurCourriel() {
    super();

    strNom = strServeur = strDomaine = "non défini";
  }

  public String zone_Identification()  {
    return CEspaceManipulation.strZoneActeurCourriel;
  }

  public CActeurCourriel(String strNom,
                         String strServeur,
                         String strDomaine) {
   super();

   this.strNom     = strNom;
   this.strServeur = strServeur;
   this.strDomaine = strDomaine;
  }


  // Représentation d'un courrier électronique dans son domaine
  public String Description() {
    return strNom + "@" + strServeur + "." + strDomaine;
  }


}
```

```
package IAMondeCommFwk.Zone.Acteur;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CActeurCourrier extends CConnaissance implements IZone {

  public CActeurCourrier() {
    super();

    this.iNumero=this.iBoite=this.iLocalite=new Integer(0);
    this.strRue = this.strPays = "non défini";
  }

  public CActeurCourrier(int iNumero,
                         int iBoite,
                         String strRue,
                         int iLocalite,
                         String strPays)
  {
    super();

    this.iNumero = new Integer(iNumero);
    this.iBoite = new Integer(iBoite);
    this.strRue = strRue;
    this.iLocalite = new Integer(iLocalite);
    this.strPays = strPays;
  }

  public String zone_Identification()  {
    return CEspaceManipulation.strZoneActeurCourrier;
  }

  // représentation d'une adresse postale dans son domaine
  public String Description() {

    return iNumero.toString()  + ", " +
           iBoite.toString()  + ", " +
           strRue + " " +
           iLocalite.toString()  + " " +
           strPays;
  }

  // --- Interface publique disponible dans l'espace
  public Integer iNumero;
  public Integer iBoite;
  public String strRue;
  public Integer iLocalite;
  public String strPays;

}
```

```
package IAMondeCommFwk .Zone.Acteur;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

import IAMondeCommFwk .*;
import IAMondeCommFwk .Espaces .*;
import IAMondeCommFwk .Zone.*;

public class CActeurHumain extends CConnaissance implements IZone {

  public CActeurHumain () {
    super();

    this.strPrenom = "non défini" ;
    this.strNom = "non défini" ;
  }

  public CActeurHumain (String strPrenom, String strNom) {
    super();

    this.strPrenom = strPrenom;
    this.strNom = strNom;
  }

  public String zone_Identification ()  {
    return CEspaceManipulation .strZoneActeurHumain ;
  }

  // construction d'un humain dans son domaine
  public String Description () {
    return strPrenom + ", "  + strNom;
  }

  // --- Interface publique disponible dans l'espace
  public String strPrenom;
  public String strNom;
}
```

```
package IAMondeCommFwk.Zone.Acteur;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CActeurNonSpecifique  extends CConnaissance implements IZone {

  public CActeurNonSpecifique() {
    super();

    this.strDescription  = "non défini";
  }

  public CActeurNonSpecifique(String strDescription) {
    super();

    this.strDescription  = strDescription;
  }

  public String zone_Identification()  {
    return CEspaceManipulation.strZoneActeurGenerique;
  }

  // représentation d'un acteur générique dans son domaine
  public String Description() {
    return this.strDescription;
  }

  // --- Interface publique disponible dans l'espace.
  public String strDescription;


}
```

```
package IAMondeCommFwk.Zone.Acteur;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CActeurTelephone  extends CConnaissance implements IZone {

  public CActeurTelephone() {
    super();

    this.strInternational = "000";
    this.strPrefixe = "00";
    this.strNumero =  "0000000";
  }

  public CActeurTelephone( String strInternational,
                           String strPrefixe,
                           String strNumero) {
    super();

    this.strInternational = strInternational;
    this.strPrefixe = strPrefixe;
    this.strNumero =  strNumero;
  }

  public String zone_Identification()  {
    return CEspaceManipulation.strZoneTelephone;
  }


  // Représentation d'un numéro de téléphone dans son domaine
  public String Description() {

    return "++ " +
           strInternational + " " +
           strPrefixe + " / " +
           strNumero;
  }

  // --- Interface publique disponible dans l'espace
  public String strInternational;
  public String strPrefixe;
  public String strNumero;
}
```

Package : Contenu

```
package IAMondeCommFwk.Zone.Contenu;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CTexteLibre extends CConnaissance implements IZone {

  // --- Interface publique disponible dans l'espace
  public String strTexte;

  public CTexteLibre() { super(); }

  public CTexteLibre(String strTexte) {
    super();
    this.strTexte = strTexte;
  }

  public String zone_Identification()  {
     return CEspaceManipulation.strZoneContenuTextLibre ;
  }
  /**
  * Lecture de la description de l'élement.
  *
  * @return description de l'élement
  */
  public String Description() {
   return "Texte au format libre" ;
  }
}
```

Package : Definition

```
package IAMondeCommFwk.Zone.Definition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */


public final class EnumDefinition {

  /**
   *  Conserve la valeur pour l'Enumération
   */
  private final long lValeur;

  // Les valeurs possible pour l'énumération
  public static final long NonDefini      = -1;
  public static final long Verbe          = 0;
  public static final long Nom            = 1;
  public static final long Adjectif       = 2;
  public static final long Adverbe        = 3;
  public static final long Abreviation    = 4;
  public static final long Preposition    = 5;
  public static final long Interjection   = 6;
  public static final long Conjonction    = 7;
  public static final long Proposition    = 8;
  public static final long Determinant    = 9;

  public static final long Phrase         = 50;
  public static final long Complement     = 51;
  public static final long Sujet          = 52;

  /** Affichage de la valeur de l'Enum */
  public Long toLong () { return new Long(lValeur); }
  public long tolong () { return lValeur; }

  public String toString() {
    switch ((int)this.tolong()) {
      case -1: return "#Non Défini#";
      case 0: return "Verbe";
      case 1: return "Nom";
      case 2: return "Adjectif";
      case 3: return "Adverbe";
      case 4: return "Abreviation";
      case 5: return "Preposition";
      case 6: return "Interjection";
      case 7: return "Conjonction";
      case 8: return "Proposition";
      case 9: return "Determinant";
      case 50: return "Phrase";
      case 51: return "Complément";
      case 52: return "Sujet";
    }
    return "#Valeur incorrecte#";
  }

  /** Unique Constructeur privé pour éviter les effets indésirable */
  public EnumDefinition (long lValeur) { this.lValeur = lValeur; }

}
```

```
package IAMondeCommFwk .Zone.Definition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public final class EnumGenre {

  /**
   *  Conserve la valeur pour l'Enumération
   */
  private final long lValeur;

  // Les valeurs possible pour l'énumération
  public static final long Masculin = 0;
  public static final long Feminin  = 1;

  /** Affichage de la valeur de l'Enum */
  public Long toLong () { return new Long(lValeur); }
  public long tolong () { return lValeur; }

  /** Unique Constructeur privé pour éviter les effets indésirable */
  public EnumGenre (long lValeur) { this.lValeur = lValeur; }

}
```

```
package IAMondeCommFwk.Zone.Definition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public final class EnumNombre implements java.io.Serializable {

  /**
   *  Conserve la valeur pour l'Enumération
   */
  private final long lValeur;

  // Les valeurs possible pour l'énumération
  public static final long Singulier = 0; // Forme au pluriel
  public static final long Pluriel   = 1;   // Forme au singulier

  /** Affichage de la valeur de l'Enum */
  public Long toLong () { return new Long(lValeur); }
  public long tolong () { return lValeur; }

  public EnumNombre (long lValeur) { this.lValeur = lValeur; }

}
```

```
package IAMondeCommFwk .Zone.Definition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */


public final class EnumPersonne implements java.io.Serializable {

  /**
   *  Conserve la valeur pour l'Enumération
   */
  private final long lValeur;

  // Les valeurs possible pour l'énumération
  public static final long  Pers_1 = 0; // première personne (je, nous)
  public static final long  Pers_2 = 1; // seconde personne  (tu, vous)
  public static final long  Pers_3 = 2; // troisième personne (il,elle,ils,elles)

  /** Affichage de la valeur de l'Enum */
  public Long toLong () { return new Long(lValeur); }
  public long tolong () { return lValeur; }

  /** Unique Constructeur privé pour éviter les effets indésirable */
  public EnumPersonne (long lValeur) { this.lValeur = lValeur; }

}
```

```
package IAMondeCommFwk.Zone.Definition;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */



public final class EnumTemps implements java.io.Serializable {

  /**
   *  Conserve la valeur pour l'Enumération
   */
  private final long lValeur;

  // Les valeurs possible pour l'énumération
  public static final long IndicatifPresent      = 0; // Indicatif présent
  public static final long IndicatifFutur        = 1; // Indicatif futur simple
  public static final long IndicatifImparfait    = 2; // Indicatif futur simple
  public static final long IndicatifPasseSimple  = 3; //Indicatif passé Simple
  public static final long SubjonctifPresent     = 4; // Subjonctif Présent
  public static final long SubjonctifImparfait   = 5; // Subjonctif Imparfait
  public static final long Infinitif             = 6;  // Infinitif
  public static final long ParticipePresent      = 7;  // Participe Present
  public static final long ImperatifPresent      = 8; // Impératif Présent
  public static final long ConditionnelPrésent   = 9;  // Conditionnel Présent
  public static final long ParticipePasse        = 10;  // Participe Passé

   /** Affichage de la valeur de l'Enum */
    public Long toLong () { return new Long(lValeur); }
    public long tolong () { return lValeur; }

    public EnumTemps (long lValeur) { this.lValeur = lValeur; }

}
```

```
package IAMondeCommFwk.Zone.Definition;

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class CDefinition  extends CConnaissance implements IZone {

  public Long lTypeDefinition=new Long(0); // type de définition : voic CDefType.
  public String strDefinition="";
  public String strNormale="";
  public Long lParam1=null;
  public Long lParam2=null;
  public Long lParam3=null;

  public CDefinition() {}


  // ----- Initialise la définition pour contenir un verbe.
  /**
   * @param strFormeConjugee
   * @param strInfinitif
   * @param enuTemps
   * @param enuNombre
   * @param enuPers
   */
  public void Verbe(String strFormeConjugee,
                    String strInfinitif,
                    EnumTemps enuTemps,
                    EnumNombre enuNombre,
                    EnumPersonne enuPers) {

    this.lTypeDefinition = new Long(EnumDefinition.Verbe);
    this.strDefinition = strFormeConjugee;
    this.strNormale = strInfinitif;
    this.lParam1 = enuTemps.toLong();
    this.lParam2 = enuNombre.toLong();
    this.lParam3 = enuPers.toLong();
  }

  public void InitVerbeFormeConjugee(String strFormeConjugee) {

    this.lTypeDefinition = new Long(EnumDefinition.Verbe);
    this.strDefinition = strFormeConjugee;
    this.strNormale = null;
    this.lParam1 = null;
    this.lParam2 = null;
    this.lParam3 = null;
  }


  public void InitGeneralFormeConjugee(String strFormeConjugee) {

    this.lTypeDefinition = null;
    this.strDefinition = strFormeConjugee;
    this.strNormale = null;
    this.lParam1 = null;
    this.lParam2 = null;
    this.lParam3 = null;
  }

  /**
   * @param strFormeConjugee
   * @param strPrimitive
   * @param enuGenre
   * @param enuNombre
```

```
 */
public void ParticipePasse (String strFormeConjugee ,
                            String strPrimitive ,
                            EnumGenre enuGenre,
                            EnumNombre enuNombre ) {

  this.lTypeDefinition     = new Long(EnumDefinition .Verbe);
  this.strDefinition  = strFormeConjugee ;
  this.strNormale     = strPrimitive ;
  this.lParam1        = enuGenre.toLong ();
  this.lParam2        = enuNombre.toLong ();
  this.lParam3        = null;
}

/**
 * @param strFormeConjugee
 * @param strPrimitive
 */
public void ParticipePresent (String strFormeConjugee ,
                              String strPrimitive ,
                              EnumGenre enuGenre,
                              EnumNombre enuNombre )  {

  this.lTypeDefinition     = new Long(EnumDefinition .Verbe);
  this.strDefinition  = strFormeConjugee ;
  this.strNormale     = strPrimitive ;
  this.lParam1        = enuGenre.toLong ();
  this.lParam2        = enuNombre.toLong ();
  this.lParam3        = null;
}

// ----- Initialise la forme infinitive d'un verbe.
/**
 * @param strInfinitif
 */
 public void Infinitif (String strInfinitif ) {
   this.lTypeDefinition  = new Long(EnumDefinition .Verbe);
   this.strDefinition    = strInfinitif ;
   this.strNormale       = strInfinitif ;
   this.lParam1          = null;
   this.lParam2          = null;
   this.lParam3          = null;
 }

 public void Generique(EnumDefinition  enuDef, String strForme )  {

  this.lTypeDefinition  = enuDef.toLong ();
  this.strDefinition    = strForme ;
  this.strNormale       = strForme ;
  this.lParam1          = null;
  this.lParam2          = null;
  this.lParam3          = null;
}

 /**
  *
  * @param strFormeConjugee
  * @param strPrimitive
  * @param enuGenre
  * @param enuNombre
  */
  public void Generique(EnumDefinition  enuDef,
                        String strFormeConjugee ,
                        String strPrimitive ,
                        EnumGenre enuGenre,
                        EnumNombre enuNombre )  {

  this.lTypeDefinition  = enuDef.toLong ();
  this.strDefinition    = strFormeConjugee ;
  this.strNormale       = strPrimitive ;
  this.lParam1          = enuGenre.toLong ();
  this.lParam2          = enuNombre.toLong ();
  this.lParam3          = null;
}
```

```
  public String zone_Identification ()  {
    return CEspaceManipulation .strZoneActeurCourriel ;
  }

  // Représentation d'une définition....
  public String toString() {
    return "";
  }


}
```

Package : Formulaires

```
package IAMondeCommFwk.Zone.Formulaires;

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;
import IAMondeCommFwk.Formulaires.*;
import java.util.*;
/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Formulaire extends CConnaissance implements IZone  {

  public String strIntitule = "";
  public Chapitres oChapitres = null;

  public Formulaire() {}

  public Formulaire(String strFormulaire, Chapitres oChapitres) {
    this.strIntitule = strIntitule;
    this.oChapitres = oChapitres;
  }

  /**
    * @return Description de la Zone
    */
  public String zone_Identification() {
    return CEspaceManipulation.strZoneFormulaire;
  }

  public boolean decodeFormulaire(String strFormulaire) {

    try {
      oChapitres.decodeFormulaire(strFormulaire);
    }
    catch (Exception e) {
      e.printStackTrace();
      return false;
    }

    return true;
}

  /**
   * Un Formulaire est valide si tous ses Chapitres sont valides
   * @return true => le Formulaire est valide, false => le Formulaire n'est pas valide.
   */
  public boolean estValide() {
    boolean fValide = true;
    Iterator chap = oChapitres.getChapitres();

    while (chap.hasNext()) {
      Chapitre c = (Chapitre) chap.next();
      if (!c.estValide()) { fValide = false; }
    }
    return fValide;
  }

  /**
   * Un Formulaire est complet si il est valide et si tous ses chapitres sont complets.
   * @return true => le Formulaire est valide, false => le Formulaire n'est pas valide.
   */
   public boolean estComplet() {
     boolean fComplet = true;

     if (! this.estValide()) { return false;}

     /** Implémentation non disponible */

     return fComplet;
```

```
    }

}
```

Package : Relations

```
package IAMondeCommFwk.Zone.Relations;

import IAMondeCommFwk.*;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class CRelation extends CConnaissance {

  // --- Interface publique disponible dans l'espace
  public EnumRelation __typeRel      = new EnumRelation(EnumRelation.NonDefini);
  public CConnaissance __Origine     = null;
  public CConnaissance __Destination = null;

  public CRelation() {
    super();
  }

  public CRelation(EnumRelation  typeRel,
                   CConnaissance __Origine,
                   CConnaissance __Destination) {
    super();

    this.__typeRel = typeRel;
    this.__Origine = __Origine;
    this.__Destination = __Destination;
  }

  public String Description() {
    return __typeRel.toString() + ": " +
           __Origine.Description() + " -> " +
           __Destination.Description();
  }

}
```

```
package IAMondeCommFwk.Zone.Relations;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class EnumRelation {

  /**
   *  Conserve la valeur pour l'Enumération
   */
  private final long lValeur;

  // Les valeurs possible pour l'énumération
  public static final long NonDefini      = -1;
  public static final long Expediteur     = 0;
  public static final long Destinataire   = 1;
  public static final long Comparable     = 2;
  public static final long Correlation    = 3;

  /** Affichage de la valeur de l'Enum */
  public Long toLong () { return new Long(lValeur); }
  public long tolong () { return lValeur; }

  public String toString() {
    switch ((int)this.tolong()) {
      case -1: return "#Non Défini#";
      case 0: return "Expéditeur";
      case 1: return "Destinataire";
      case 2: return "Comparable";
      case 3: return "Correlation";
    }
    return "#Valeur incorrecte#";
  }

  /** Unique Constructeur privé pour éviter les effets indésirable */
  public EnumRelation (long lValeur) { this.lValeur = lValeur; }
}
```

Package : Synonymes

```
package IAMondeCommFwk.Zone.Synonymes;

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;
import IAMondeCommFwk.Formulaires.*;
import java.util.*;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public class Synonyme extends CConnaissance implements IZone  {


  public String strMot_0 = ""; // mot synonyme mot0 voir spec.analyseur.10
  public String strMot_n = ""; // Mot de Base mot1..motn

  public Synonyme() {}

  public Synonyme(String strMot_0, String strMot_n) {
    this.strMot_0 = strMot_0;
    this.strMot_n = strMot_n;
  }

  /**
    * @return Description de la Zone
    */
  public String zone_Identification()  {
    return CEspaceManipulation.strZoneSynonyme;
  }

}
```

Package : Validations

```
package IAMondeCommFwk.Zone.Validations;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CEstPasVide extends CConnaissance implements IValidations, IZone {

  public CEstPasVide() { }

  /**
    * @return Description de la Zone
    */
  public String zone_Identification()  {
    return CEspaceManipulation.strZoneValidationEstPasVide;
  }

  /**
   * @param strContenu
   *   la valeur de l'élément à valider.
   *
   * @return
   *   true: l'élément est non vide.
   *   false: l'élément est vide.
   */
  public boolean Validation(String strContenu) {
    return strContenu.length() > 0 ? true : false;
  }
}
```

```
package IAMondeCommFwk.Zone.Validations;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */
import java.text.*;

import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

/**
 *  Permet de validaer une date, le modèlme peut être étendu
 *  pour valider différents type de date....
 */
public class CEstUneDate  extends CConnaissance implements IValidations, IZone {

  public CEstUneDate() {}

  /**
    * @return Description de la Zone
    */
  public String zone_Identification()  {
    return CEspaceManipulation.strZoneValidationEstDate;
  }

  /**
   * @param strContenu
   *   la valeur de l'élément à valider.
   *
   * @return
   *   true: l'élément est une date valide.
   *   false: l'élément n'est pas une date.
   */
  public boolean Validation(String strContenu) {

    // --- Si l'élément n'est pas remplis, alors il est une date valide
    if (0 == strContenu.length()) {return true;}

    SimpleDateFormat date = new SimpleDateFormat();

    try {
      date.parse(strContenu);
    } catch (ParseException e) { return false; }

    return true;
  }
}
```

```
package IAMondeCommFwk.Zone.Validations;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */
import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CEstUnNombre  extends CConnaissance implements IValidations, IZone {

  public CEstUnNombre() {}

  /**
    * @return Description de la Zone
    */
  public String zone_Identification()  {
   return CEspaceManipulation.strZoneValidationEstUnNombre;
  }

  /**
   * @param strContenu
   *   la valeur de l'élément à valider.
   *
   * @return
   *   true: l'élément est un nombre valide.
   *   false: l'élément n'est pas un nombre valide.
   */
  public boolean Validation(String strContenu) {
    if ( strContenu == String.valueOf(strContenu).toString() ) {
      return true;
    }
    else { return false; }
  }
}
```

```
package IAMondeCommFwk.Zone.Validations;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */
import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CEstUnNombre  extends CConnaissance implements IValidations, IZone {

  public CEstUnNombre() {}

  /**
    * @return Description de la Zone
    */
  public String zone_Identification()  {
   return CEspaceManipulation.strZoneValidationEstUnNombre;
  }

  /**
   * @param strContenu
   *   la valeur de l'élément à valider.
   *
   * @return
   *   true: l'élément est un nombre valide.
   *   false: l'élément n'est pas un nombre valide.
   */
  public boolean Validation(String strContenu) {
    if ( strContenu == String.valueOf(strContenu).toString() ) {
      return true;
    }
    else { return false; }
  }
}
```

```
package IAMondeCommFwk.Zone.Validations;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */
import IAMondeCommFwk.*;
import IAMondeCommFwk.Espaces.*;
import IAMondeCommFwk.Zone.*;

public class CTailleMax  extends CConnaissance implements IValidations, IZone {

  private long lTailleMax = 0;

  public CTailleMax() {}

  public CTailleMax(long lTailleMax) {
    this.lTailleMax = lTailleMax;
  }

 /**
   * @return Description de la Zone
   */
  public String zone_Identification()  {
    return CEspaceManipulation.strZoneValidationTailleMax  +
           " #" + String.valueOf(lTailleMax) + "#";
  }

 /**
   * @param strContenu
   *   la valeur de l'élément à valider.
   *
   * @return
   *   true: l'élément est de taille maximale lTailleMax
   *   false: l'élément est de taille supérieurs à lTailleMax
   */
  public boolean Validation(String strContenu) {
    if ( strContenu.length() <= lTailleMax ) {
      return true;
    }
    else { return false; }
  }
}
```

```
package IAMondeCommFwk.Zone.Validations;

/**
 * <p>Title: IA Mailer Framework</p>
 * <p>Description: Mémoire Philippe Verdeyen</p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: </p>
 * @author Philippe Verdeyen
 * @version 1.0
 */

public interface IValidations  {
  boolean Validation(String strContenu);
}
```

Références

[Grevisse] **Précis de grammaire française.**
Collection Grevisse
Maurice Grevisse
Editions J.Duculot s.a., Gembloux

Applying Natural Language Processing (NLP) Based Metadata Extraction to Automatically Acquire User Preferences.
Woojin Paik, Sibel Yilmazel, Eric Brown,
Maryjane Poulin, Stephane Dubon et Christophe Amice
ACM Digital library

Representation of legal text for conceptual retrieval.
Judith P.Dick
ACM Digital library

Improving the Representation of Legal Case Texts with Information Extraction Methods.
Stefanie Brüninghaus et Kevin D. Ashley
Learning Research and Development Center
University of Pittsburgh, Pittsburgh PA 15260

Designing text retrieval systems for 'Conceptual Searching'.
Jon Bing
Norwegian Research Center for Computers and Law
University of Oslo
Norway

How to write parallel programs (a first course).
Nicholas Carriero – David Gelernter
The MIT Press

Javaspaces Principles, Patterns, and Practice.
Freeman – Hupfer – Arnold
ISBN: 0-201-30955-6
JavaSpaces in practice.
Philip Bishop et Nigel Warren
ISBN: 0-321-11231-8
Make Room for JavaSpaces.
Eric Freeman
http://www.javasoft.com/javaspaces
http://www.jini.org
http://www.gigaspaces.com

Références préparatoires

Voici la liste des références utilisées pour l'étude des méthodologies existantes dans le monde de la gestion de l'information. Ces références ne sont pas reprises dans la rédaction du mémoire.

Query Reformulation for Dynamic Information Integration.
Yigal Arens, Craig A. Knoblock, Wei-Min Shen
Information sciences Institute and Department of Computer Science
University of Southern California

LDC-1 : A transportable, Knowledge-Based Natural Language Processor for Office Environments.
Bruce W. Ballard, John C. Lusth et Nancy L. Tinkham
Duke University

Living with CLASSIC : When and How to Use a KL-ONE-Like Language.
Ronal J Brachman, Deborah L. McGuinness
Peter F. Patel-Schneider et Lori Alperin Resnick
San Mateo, CA, 1991

CLASSIC Learning.
Michael Frazier, Leonard Pitt
Editor : Tom Hancock

A situated Classification Solution of a Resource Allocation Task Represented in a Visual Language.
Brian R Gaines
Knowledge Science Institute, University of Calgary

Knowledge Interchange Format.
draft proposed American National Standard (dpANS)
NCITS.T2/98-004.

Towards a Method to Conceptualize Domain Ontologies.
Asunción Gómez-Pérez, Marianon Fernández, Antonio J. de Vicente
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid

Computational Intelligence, a logical approach.
David Poole, Alan Mackworth et Randy Goebel
University of British Columbia
ISBN : 0-19-510270-3

Supporting Ontological Analysis of Taxonomic Relationships.
Cristopher Welty, Nicola Guarino
Vassar College, Poughkeepsie, USA LADSEB-CNR, Padova, Italy